

Dpto Sistemas Electrónicos y de Control
Universidad Politécnica de Madrid

Embedded Linux Systems

Using Buildroot for building Embedded Linux
Systems with the Raspberry-PI
V1.2

Mariano Ruiz

2014



Table of contents

1	SCOPE.....	6
1.1	Document Overview	6
1.2	Acronyms	6
2	REFERENCED DOCUMENTS	7
2.1	References.....	7
3	LAB1: BUILDING LINUX USING BUILDROOT	8
3.1	Elements needed for the execution of these LABS.....	8
3.2	Starting the VMware.....	8
3.3	Configuring Buildroot.....	11
3.4	Compiling buildroot.	15
3.5	Buildroot Output.....	15
3.6	Configuring the Linux kernel parameters	16
3.7	Booting the Raspberry Pi.....	21
3.8	Booting the RaspBerry Pi using a script.	26
3.9	Configuring the network interface	26
4	LAB2: CROSS-COMPILING APPLICATIONS FOR BEAGLEBOARD	27
4.1	Hello Word application.....	27
5	LAB3: USING INTEGRATED DEVELOPMENT ENVIRONMENT: ECLIPSE/CDT	30
5.1	Cross-Compiling application using Eclipse.	30
6	PREPARING THE LINUX VIRTUAL MACHINE.	38
6.1	Download VMware Player.....	38
6.2	Installing Ubuntu 12.04 LTS as virtual machine.....	38
6.3	Installing packages for supporting Buildroot.	38
7	ANNEX I: UBUNTU 12.04 LTS PACKAGES INSTALLED.	40
7.1	List of packages installed in the Ubuntu OS.....	40
7.2	Installing software in Ubuntu 12.04 TLS	40

Table of figures

Fig. 1: Main screen of VMware player with some VM available to be executed.	8
Fig. 2: Ubuntu Virtual Machine login screen.	9
Fig. 3 Buildroot home page.	9
Fig. 4: Downloading buildroot source code.	10
Fig. 5: Buildroot folder (the folder name depends on the version doanloaded).	10
Fig. 6: Buildroot setup screen.	11
Fig. 7: Successful compilation and installation of Buildroot	15
Fig. 8: Schematic representation of the Buildroot tool. Buildroot generates the root file system, the kernel image, the bootloader and the toolchain. Figure copied from "Free Electrons" training materials (http://free-electrons.com/training/)	16
Fig. 9: images folder contains the binary files for our embedded system.	16
Fig. 10: Main window for configuring the Linux kernel.	17
Fig. 11: Selection of support for initial ram disk.	18
Fig. 12: Removing the support for MMC/SD/SDIO in the Linux Kernel.	19
Fig. 13: Selection of the "extend bootloader kernel arguments" option.	20
Fig. 14: Default argument to add.	20
Fig. 15: RaspBerry-Pi (Version B) hardware with main elements identified.	21
Fig. 16: RaspBerry-Pi header terminal identification.	22
Fig. 17: Identification of the terminals in the USB-RS232 adapter	22
Fig. 18: Booting process for BMC2835 processor in the rasperry-pi	23
Fig. 19: Putty program main window.	24
Fig. 20: Serial output in the Raspberry Pi boot process..	24
Fig. 21: Beagle running Linux.	26
Fig. 22: Basic hello world program in C.	27
Fig. 23: Summary of the different configurations for developing applications for embedded systems. Figure copied from "Free Electrons" training materials (http://free-electrons.com/training/)	27
Fig. 24: Folder containing the cross compiling tools.	28
Fig. 25: Adding the cross compiler PATH to the Linux .profile file.	29
Fig. 33: Selection of the workspace for Eclipse.	30
Fig. 34: Eclipse welcome window.	31
Fig. 35: Eclipse main window.	31
Fig. 36: Creation of the hello world C project.	32
Fig. 37: Hello world example.	32
Fig. 38: Tool Chain Editor should be configured to use Cross GCC.	33
Fig. 39: Cross tools locate on (path)	33
Fig. 40: Include search path.	34
Fig. 41: Libraries search path.	34
Fig. 42: Eclipse project compiled (Binaries has been generated).	35
Fig. 43: Creating a Debug Configuration	35
Fig. 44: Configuration must be set to manual remote debugging.	36
Fig. 45: Debug configuration including the path to locate the cross gdb tool.	36
Fig. 46: Configuration of the remote target. Port number must be the same in the target and in the host.	37

1 SCOPE

1.1 Document Overview

- This document describes the basic steps to develop and embed a Linux-based system using the Raspberry PI board. The document has been specifically written to use a Raspberry-PI development system based on the BCM2835 processor. All the software elements used have a GPL license.



[Time to complete the tutorial]: The time necessary to complete all the steps in this tutorial is approximately 8 hours.

Read carefully all the instructions before executing the practical part otherwise you will find errors and probably unpredicted errors. In parallel you need to review the slides available at [UPM ISE moodle](#) site or at [RD1]

1.2 Acronyms

CPU	Central Processing Unit
EABI	Embedded-Application Binary Interface
EHCI	Enhanced Host Controller Interface
I/O	Input and Output
MMC	Multimedia card
NAND	Flash memory type for fast sequential read and write
PCI	Peripheral Component Interconnect – computer bus standard
PCI Express	Peripheral Component Interconnect Express
OS	Operating system
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus

2 REFERENCED DOCUMENTS

2.1 References

- [RD1] Embedded Linux system development. Slides at <http://free-electrons.com/doc/training/embedded-linux/>
- [RD2] Hallinan, C. Embedded Linux Primer. Second Edition. Prentice Hall. 2011.
- [RD3] [getting-started-with-ubuntu](#)
- [RD4] <http://free-electrons.com/training/embedded-linux/>
- [RD5] Raspberry-Pi User Guide. Reference Manual. [www.myraspberrypi.org/wp-content/.../Raspberry.Pi .User .Guide .pdf](http://www.myraspberrypi.org/wp-content/uploads/2015/04/Raspberry-Pi-User-Guide.pdf)
- [RD6] <http://www.uclibc.org/> uclib web site.
- [RD7] <http://www.gnu.org/software/binutils/> Binutils web site.

3 LAB1: BUILDING LINUX USING BUILDROOT

3.1 Elements needed for the execution of these LABS.

In order to execute properly this lab you need the following elements:

1. WMWare player version 5.0 or above. Available at www.wmware.com (free download and use). This software is already installed in the laboratory desktop computer.
2. A VMWare virtual machine with Ubuntu 12.04 and all the software packages installed is already available in the Desktop. This virtual machine is available for your personal use at the Department assistance office (preferred method). If you want to setup your virtual machine by yourself follow the instructions provided in the Annex I.
3. A Raspberry-Pi with the USB cable is available at ISE laboratory.

3.2 Starting the VMware

Start WMware player and open the ISE Virtual Machine. Wait until the welcome screen is displayed (see Fig. 1 and Fig. 2). Login as "ISE" user using the password "ise.2013". A Ubuntu tutorial is available at moodle site.

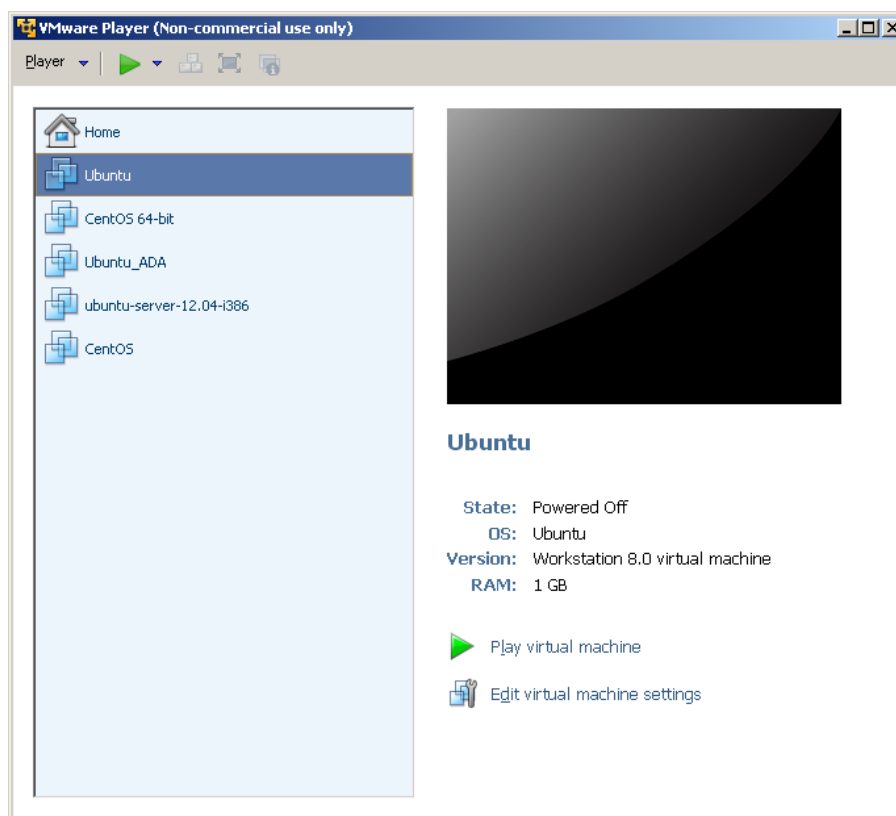


Fig. 1: Main screen of VMware player with some VM available to be executed.

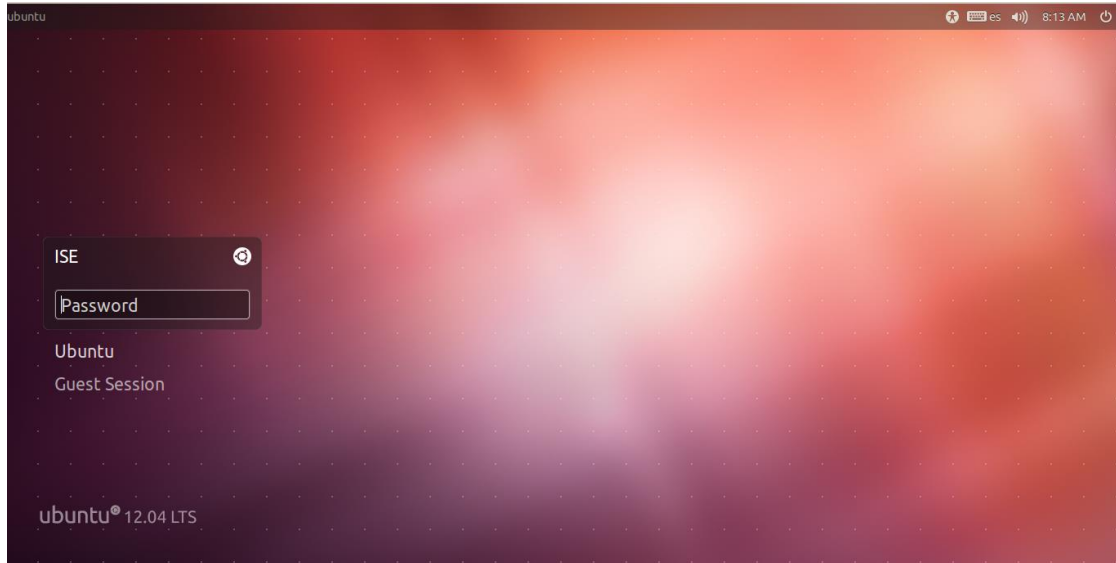


Fig. 2: Ubuntu Virtual Machine login screen.

Open the **firefox** web browser and download from <http://buildroot.uclibc.org/> the version identified as buildroot2013-11 (use the download link, see Fig. 3, and navigate searching for earlier releases). Save the file to the **Documents** folder in your account (Fig. 4).

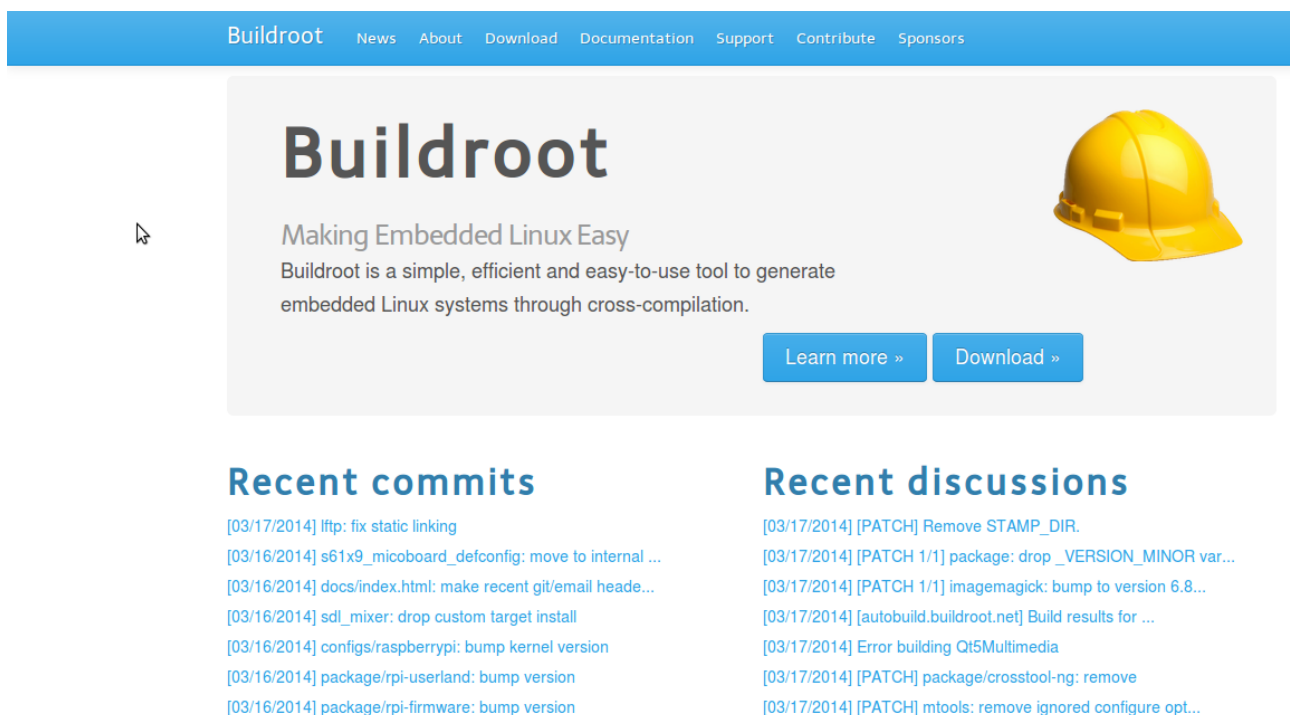


Fig. 3 Buildroot home page.

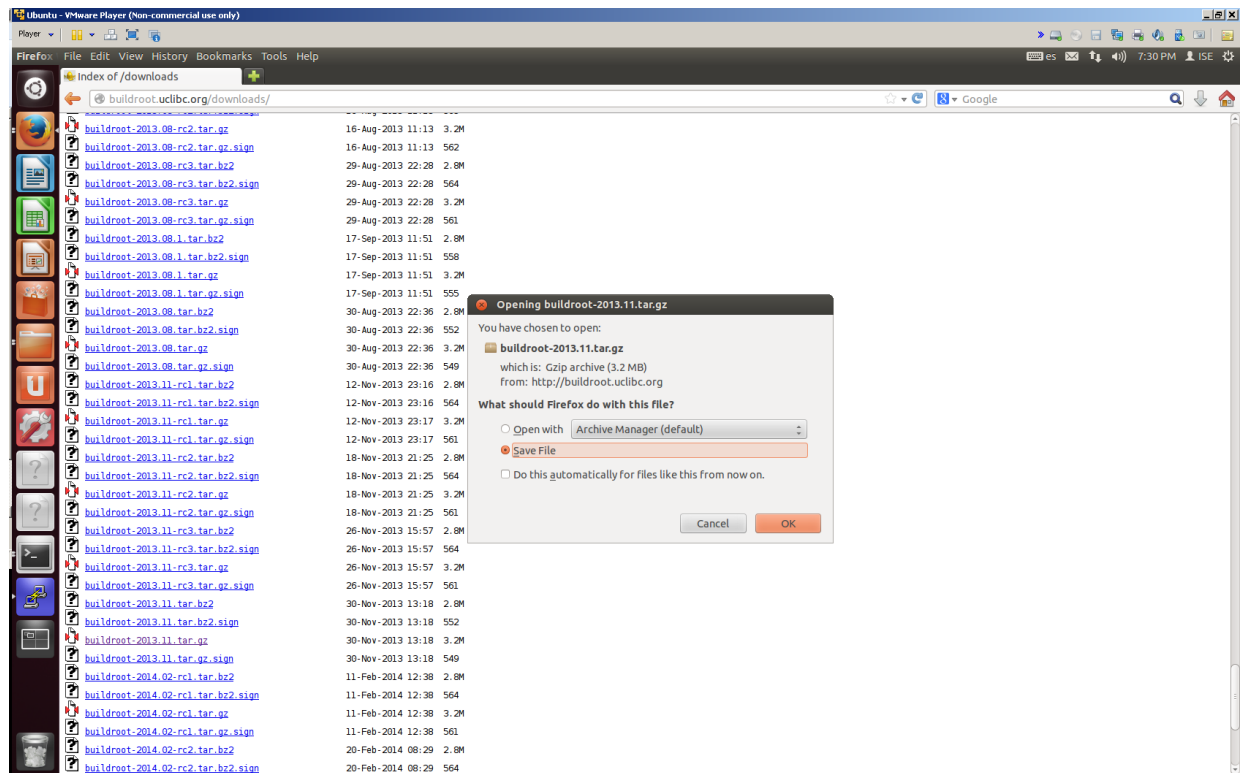


Fig. 4: Downloading buildroot source code.

Using the file explorer navigate to the Documents folder and decompress the file (Fig. 5).

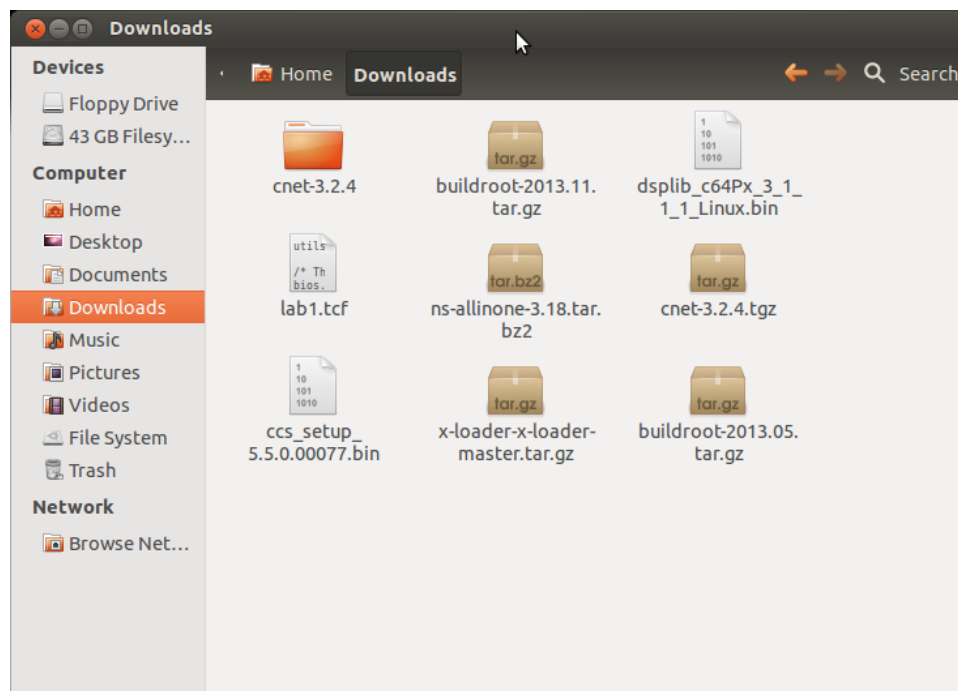


Fig. 5: Buildroot folder (the folder name depends on the version doanloaded).

Right click in the window and execute "Open in Terminal". In some seconds a command window is displayed. Then, execute these commands:

```
ise@ubuntu:~/Documents$ cd buildroot-2013.11
ise@ubuntu:~/Documents/buildroot-2013.11$ make xconfig
```



[Help]: In Linux “TAB” key helps you to autocomplete the commands, folders and files names.

In some seconds you will see a new window similar to Fig. 6.

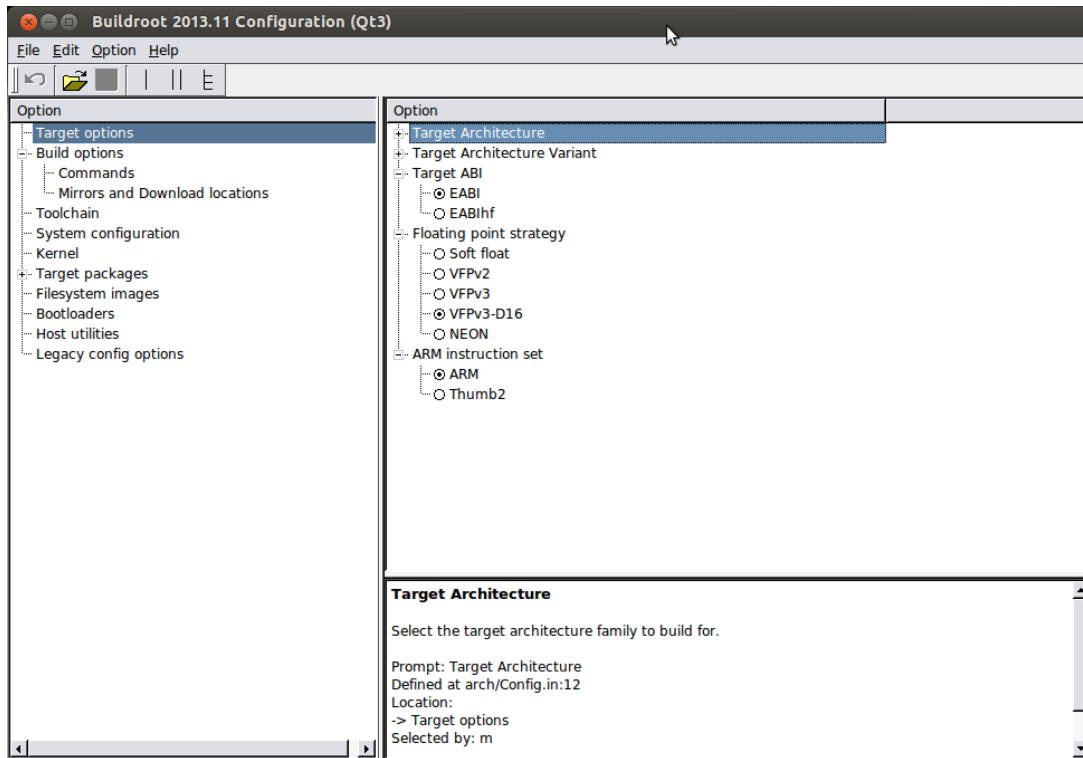


Fig. 6: Buildroot setup screen.

3.3 Configuring Buildroot.

Once **Buildroot** configuration is started, it is necessary to configure the different items. You need to navigate through the different menus and select the elements to install. Table I contains the specific configuration of **Buildroot** for installing it in the Raspberry Pi. Depending on the version downloaded the organization and the items displayed can be different.



[Help]: The Buildroot configuration is an iterative process. In order to set up your embedded Linux system you will need to execute the configuration several times.

Table I: Parameters for Buildroot configuration

Main Item	Subitem	Value	Comments
Target options	Target Architecture	ARM (little endian)	
	Target Architecture Variant	Arm1176jzf-s	
	Target ABI	EABI	An embedded-application binary interface (EABI) specifies standard conventions for file formats,

Main Item	Subitem	Value	Comments
			data types, register usage, stack frame organization, and function parameter passing of an embedded software program.
	Floating point strategy	VFPv2	
	ARM instruction Set	ARM	
Build options		Default values	How Buildroot will built the code. Leave default values.
Toolchain			Cross Compiler, linker, libraries to be built to compile out embedded application
	Toolchain Type	Buildroot toolchain	Embedded system will be compiled with tools integrated in Buildroot
	Kernel Header s	Manually specified Linux linux version: 3.6.11	Source header files of the Linux Kernel.
	C library	uClibc	
	uClibc C library Version	Version 0.9.33.x	Library (small size version) containing the typical C libraries used in Linux environments (stdlib, stdio, etc)[RD6]
	uClibc configuration file to use?	package/uClibc/0.9.33.config	
	Enable large file (files > 2GB) support	Yes	
	Enable WCHAR support	Yes	Support for extender set of chars.
	Thread library implementation	Native POSIX Threading (NPTL)	
	Thread library debugging	Yes	Embedded system will have debugable threads,
	Compile and install uClibc utilities	Yes	
	Binutils Version	binutils 2.22	Leave default values [RD7]. Binutils contains tools to manage the binary files obtained in the compilation of the different applications
	GCC compiler Version	gcc 4.7.x	GCC tools version to be installed
	Enable C++ support		Including support for C++ programming, compiling, and linking.
	Build cross gdb for the host	gdb 7.5.x	Includes the support for GDB.

Main Item	Subitem	Value	Comments
			GCC debugger.
	Enable MMU support		Mandatory if building a Linux system
	Target Optimizations:	-pipe	
System Configuration			
	System Hostname	buildroot	Name of the embedded system
	System Banner	Linux ISE	Banner
	Passwords encoding	md5	
	Init System	Busybox	
	/dev management	Dinamic using devtmpfs only	
	Path to permissions table	system/device_table.txt	Text files with permissions for /dev files
	Root filesystem skeleton	Default target skeleton	Linux folder organization for the embedded system
	Root password	ise	
	Run a getty: Port to run a getty	/dev/tty1 /dev/ttyAMA0	Linux device file with the port to run getty (login) process. Uses ttyAMA0 for serial port
	Baud rate to use	115200	
	remount root filesystem read-write during boot	Yes	
Linux Kernel			
	Kernel version	Custom Git Repository	
	URL of custom repository	git://github.com/raspberrypi/linux.git	
	Custom repository version	1587f77	
	Kernel configuration	Using defconfig	
	Defconfig name	bcmrpi_quick	
	Kernel binary format	ulmage	
	Linux Kernel Extensions	Nothing	
Target Packages			
	Busybox	BusyBox 1.21.x	
	Busybox configuration file to use	package/busybox/busybox-1.21.x.config	
	Audio and video applications	Default values	
	Compressors and decompressors	Default values	
	Debugging, profiling and	Gdb, gdbserver	

Main Item	Subitem	Value	Comments
	benchmark		
	Developments tools	Default values	
	Filesystem and flash utilities	Default values	
	Games	Default values	
	Graphic libraries and applications (graphic/text)	Default values	
	Hardware handling	Firmware->rpi-firmware	
	Interpreters language and scripting	Default values	
	Libraries	Other->libcofi	
	Miscellaneous Networking applications Package managers Real Time Shell and utilities System Tools Text Editors and viewers	Default	
Filesystem Images			
	ext2/3/4 root filesystem	ext2 (rev0) size 16384 Compression method gzip	
	tar the root filesystem	no compression	
Bootloaders			
	U-Boot	U-Boot board name: rpi_b U-Boot Version: 2013.10 U-Boot binary format: uboot.bin	
Host utilities			
	host u-boot tools	Yes	
Legacy config options		Default values	

Once you have configured all the menus you need to exit saving the values (File->Quit).



[Help]: The **Buildroot** configuration is stored in a file named as “.config”. You should have a backup of this file.

3.4 Compiling buildroot.

In the Terminal Window executes the following command:

```
ise@ubuntu:~/Documents/buildroot-2013.11$ make
```

If everything is correct you will see a final window similar to the represented in Fig. 7.



[Time for this step]: In this step buildroot is going to connect, using internet, to different repositories. After downloading the code, Buildroot is going to compile the applications and generates a lot files and folders. Depending of your internet speed access and the configuration chosen this step could take up to **one hour**.

```
ise@ubuntu: ~/Documents/buildroot-2013.11
cuments/buildroot-2013.11/output/host/usr/bin:/home/ise/Documents/buildroot-2013
.11/output/host/usr/sbin:/home/ise/Documents/buildroot-2012.02-rc1/output/host/
usr/bin:/usr/lib/lightdm/lightdm:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/b
in:/sbin:/bin:/usr/games" GEN=2 REV=0 fs/ext2/genext2fs.sh -d /home/ise/Document
s/buildroot-2013.11/output/target -b 64000 /home/ise/Documents/buildroot-2013.1
1/output/images/rootfs.ext2" >> /home/ise/Documents/buildroot-2013.11/output/bui
ld/_fakeroot.fs
chmod a+x /home/ise/Documents/buildroot-2013.11/output/build/_fakeroot.fs
/home/ise/Documents/buildroot-2013.11/output/host/usr/bin/fakeroot -- /home/ise/
Documents/buildroot-2013.11/output/build/_fakeroot.fs
rootdir=/home/ise/Documents/buildroot-2013.11/output/target
table='/home/ise/Documents/buildroot-2013.11/output/build/_device_table.txt'
tune2fs 1.42.8 (20-Jun-2013)

e2fsck was successfully run on 'rootfs.ext2' (ext2)

tune2fs 1.42.8 (20-Jun-2013)
Setting maximal mount count to -1
Setting interval between checks to 0 seconds
cp support/misc/target-dir-warning.txt /home/ise/Documents/buildroot-2013.11/out
put/target/THIS_IS_NOT_YOUR_ROOT_FILESYSTEM
gzip -9 -c /home/ise/Documents/buildroot-2013.11/output/images/rootfs.ext2 > /ho
me/ise/Documents/buildroot-2013.11/output/images/rootfs.ext2.gz
ise@ubuntu:~/Documents/buildroot-2013.11$
```

Fig. 7: Successful compilation and installation of Buildroot

Buildroot has generated some folders with different files and subfolders containing the tools for generating your Embedded Linux System. Next paragraph explains the main outputs obtained,

3.5 Buildroot Output.

The main output files of the execution of the previous steps can be located at the folder `"/output/images"`. Fig. 8 summarizes the use of **Buildroot**. Buildroot generates a boot loader, a kernel image, and a file system.

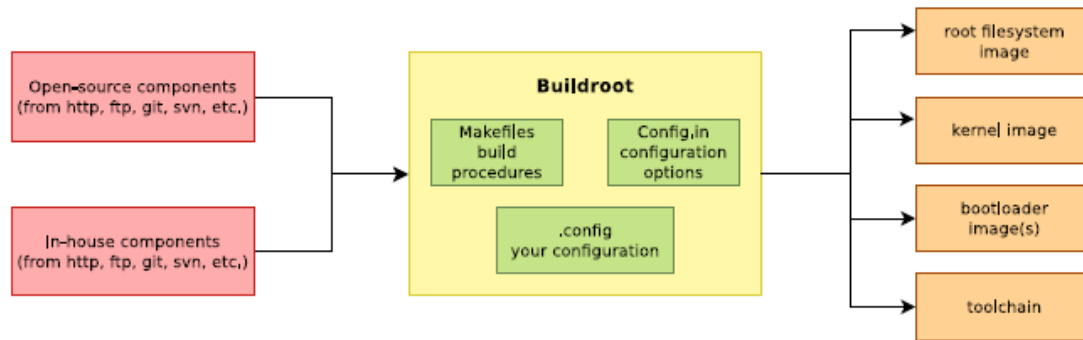


Fig. 8: Schematic representation of the Buildroot tool. Buildroot generates the root file system, the kernel image, the bootloader and the toolchain. Figure copied from “Free Electrons” training materials (<http://free-electrons.com/training/>)

In our specific case the folder content is shown in Fig. 9

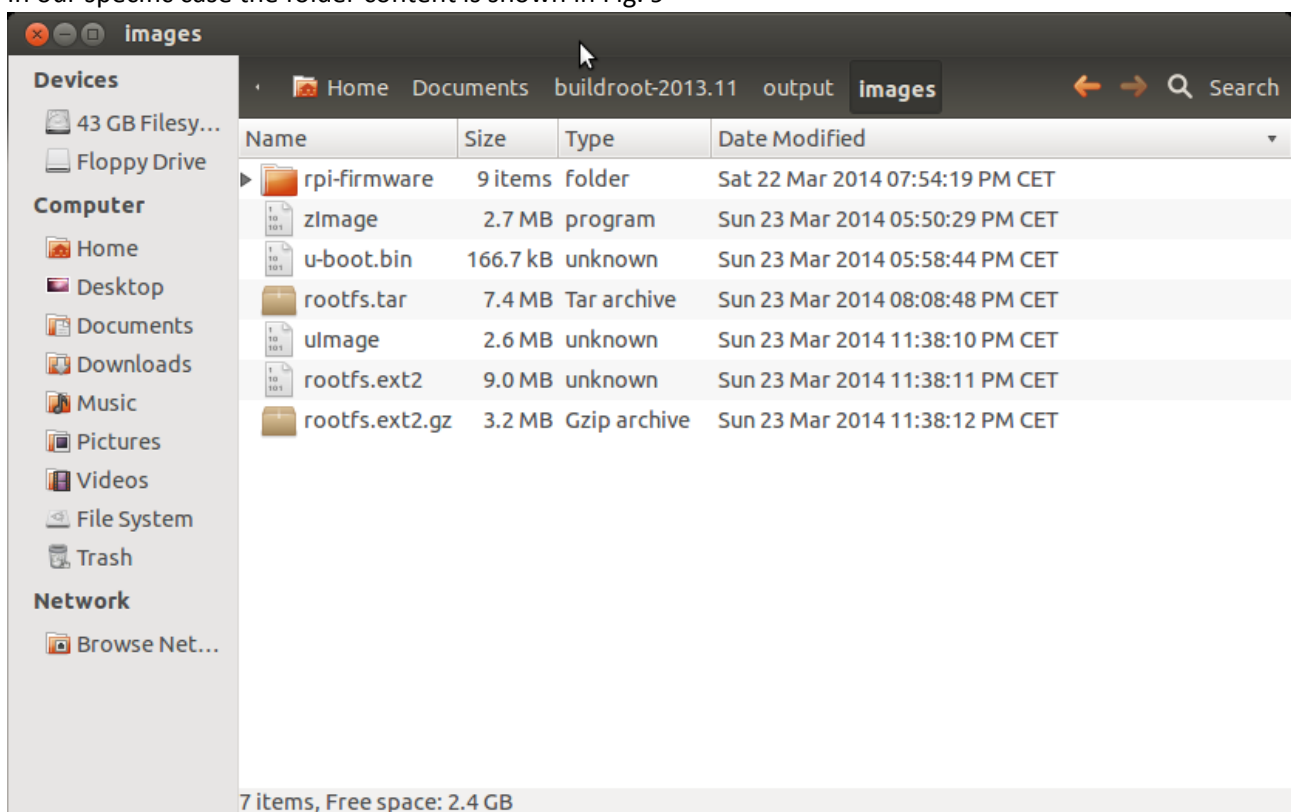


Fig. 9: images folder contains the binary files for our embedded system.

These files must be copied to a SD card for booting Raspberry Pi. The u-boot.bin file contains the u-boot application for booting your Linux. Linux image is ulmage and rootfs.ext2.gz contains the file system compressed. The file rootfs.ext2 is the decompressed version of rootfs.ext2.gz and is not necessary for booting. The folder rpi-firmware contains additional mandatory files to boot the raspberry. The source code of this firmware is closed by the manufacturer.

3.6 Configuring the Linux kernel parameters

In all the Linux embedded applications it is necessary to set up the kernel in order to support the different physical devices and user space applications. All kernel sources provide a basic configuration for specific hardware platforms. If you are using a commercial hardware platform probably you will find a kernel configuration suitable for you, if not you will need to define it. This is a hard task but you will find hundreds

of examples in the Internet. Linux kernel has a directory with predefined hardware configurations. The relative path is this: “./Documents/buildroot-2013.11/output/build/linux-1587f77/arch/arm/configs”. Have a look to this folder and you will find this file: *bcmrpi_defconfig*.

Executing the command:

```
ise@ubuntu:~/Documents/buildroot-2013.11$ make linux-menuconfig
```

you can navigate in this application) using the arrows. Inspect the configuration of the kernel. If you add or remove features. Save it and compile again buildroot using make. **These will generate new images that must be copied to SD card.** In order to boot the raspberry-pi without errors you need to modify the kernel configuration.

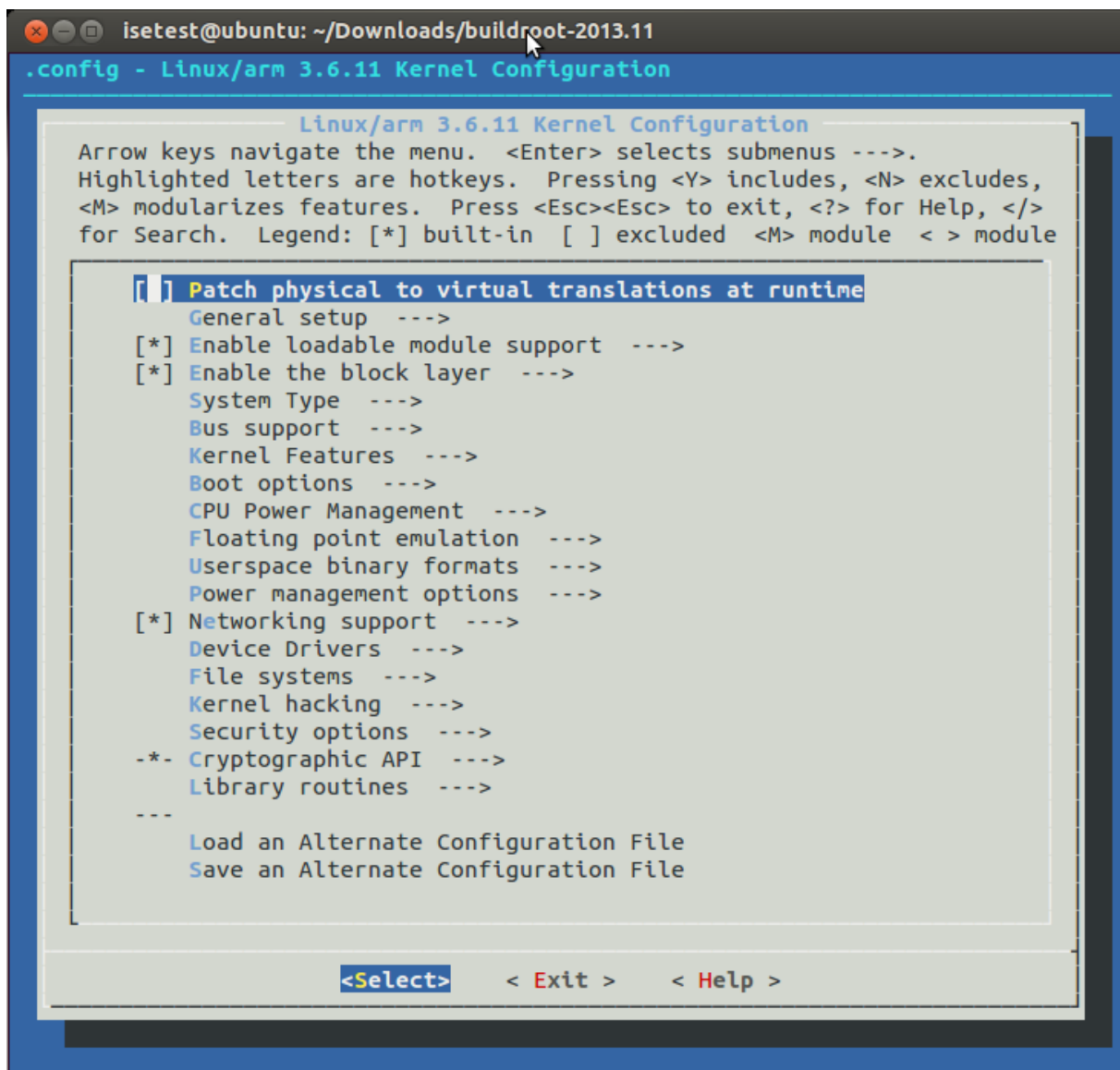


Fig. 10: Main window for configuring the Linux kernel.

Now select “General Setup” and select the options “Initial RAM filesystem and RAM disk (initramfs/initrd) support” and “Support initial ramdisks compressed using gzip” (see Fig. 11).

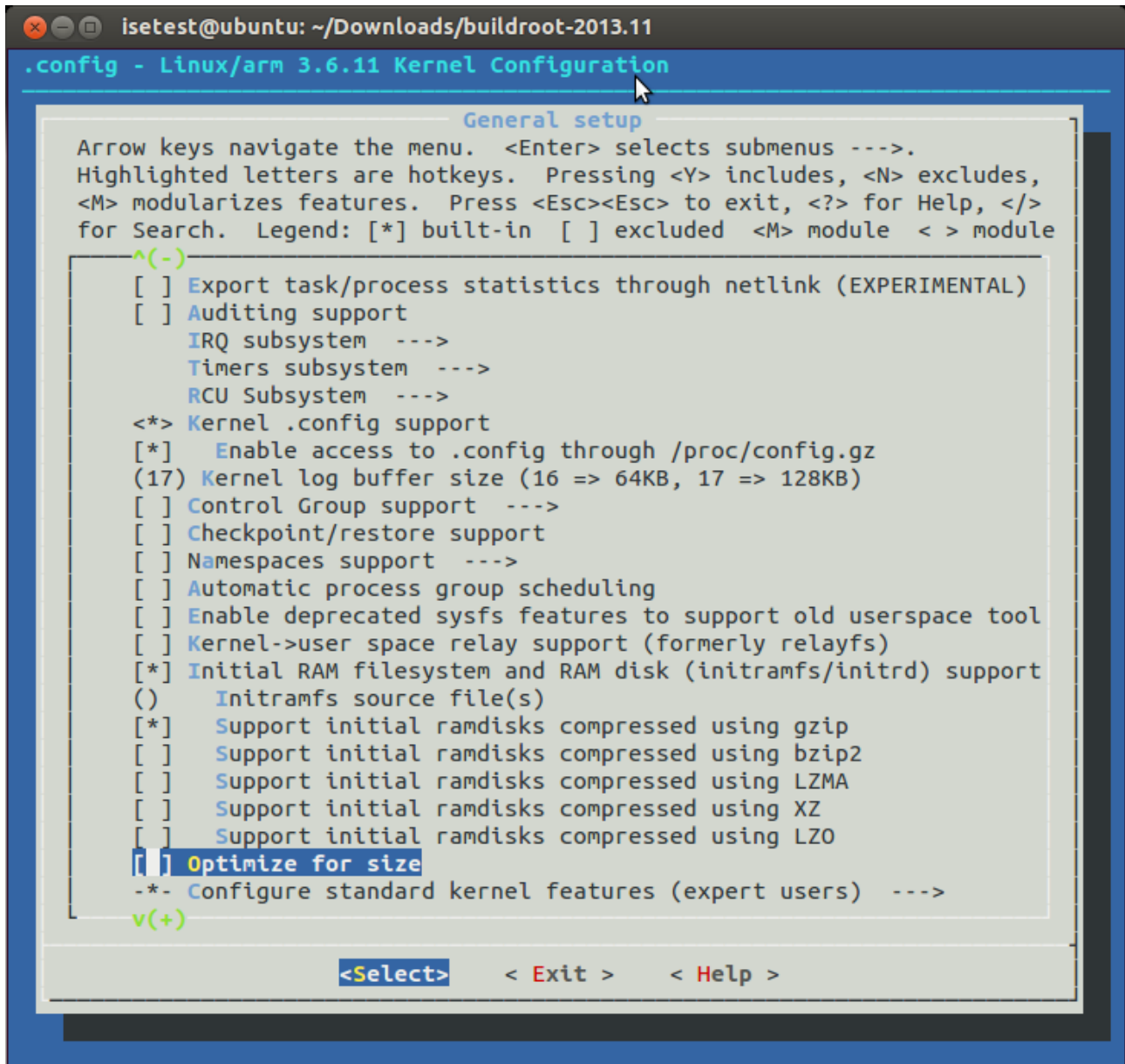


Fig. 11: Selection of support for initial ram disk.

Now access to Device Drivers and deactivated the use of MMC/SD/SDIO cards drivers are shown in Fig. 12

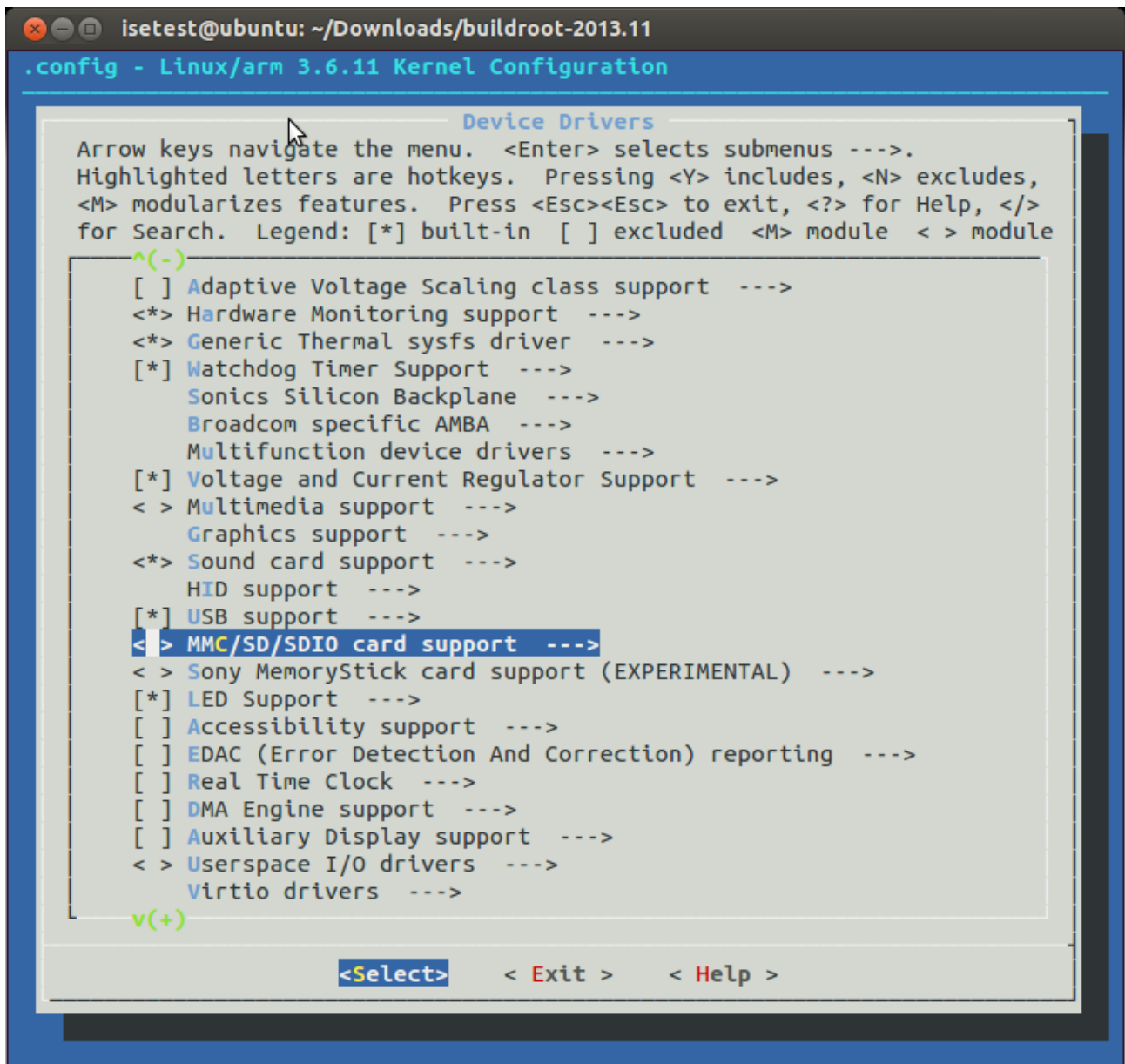


Fig. 12: Removing the support for MMC/SD/SDIO in the Linux Kernel.

The last step is to configure the additional boot parameters for the kernel. Select kernel boot options->kernel command line type and select "extend bootloaders kernel arguments" (see Fig. 13). Now in the default kernel command string insert "fiq_fix_enable=1 sdhci-bcm2708.sync_after_dma=0 dwc_otg.lpm_enable=0".

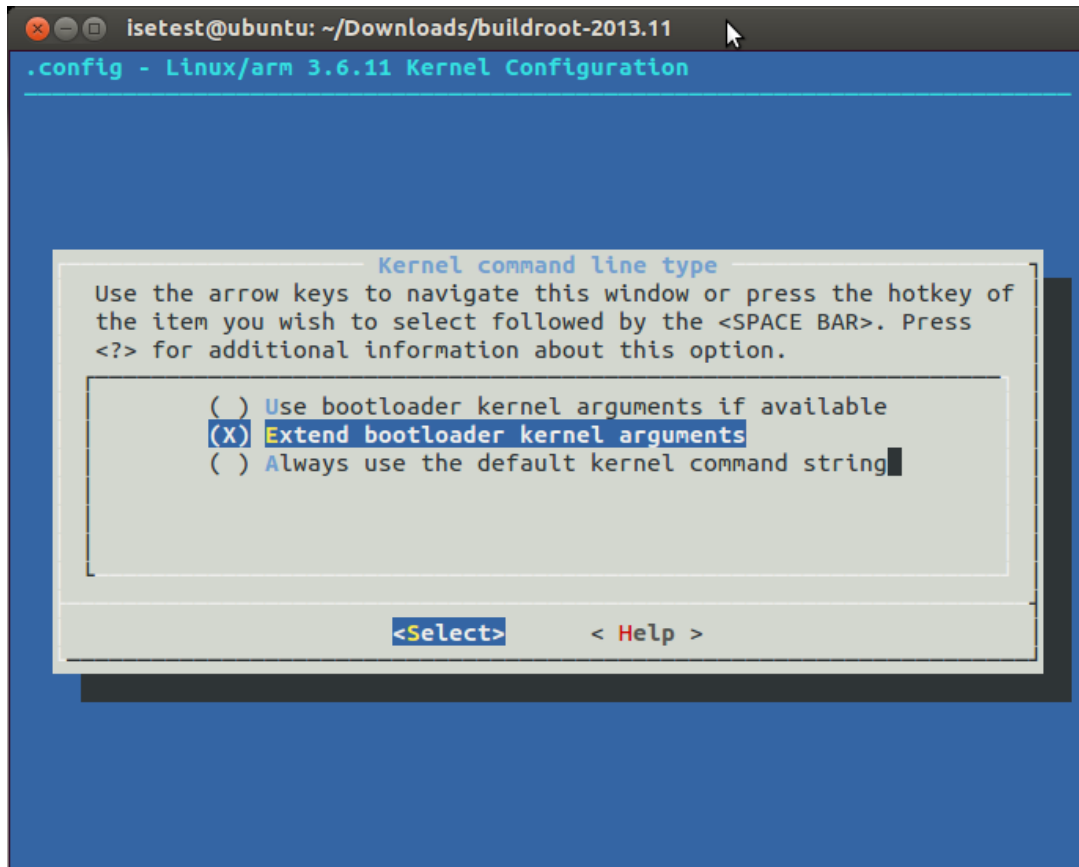


Fig. 13: Selection of the “extend bootloader kernel arguments” option.

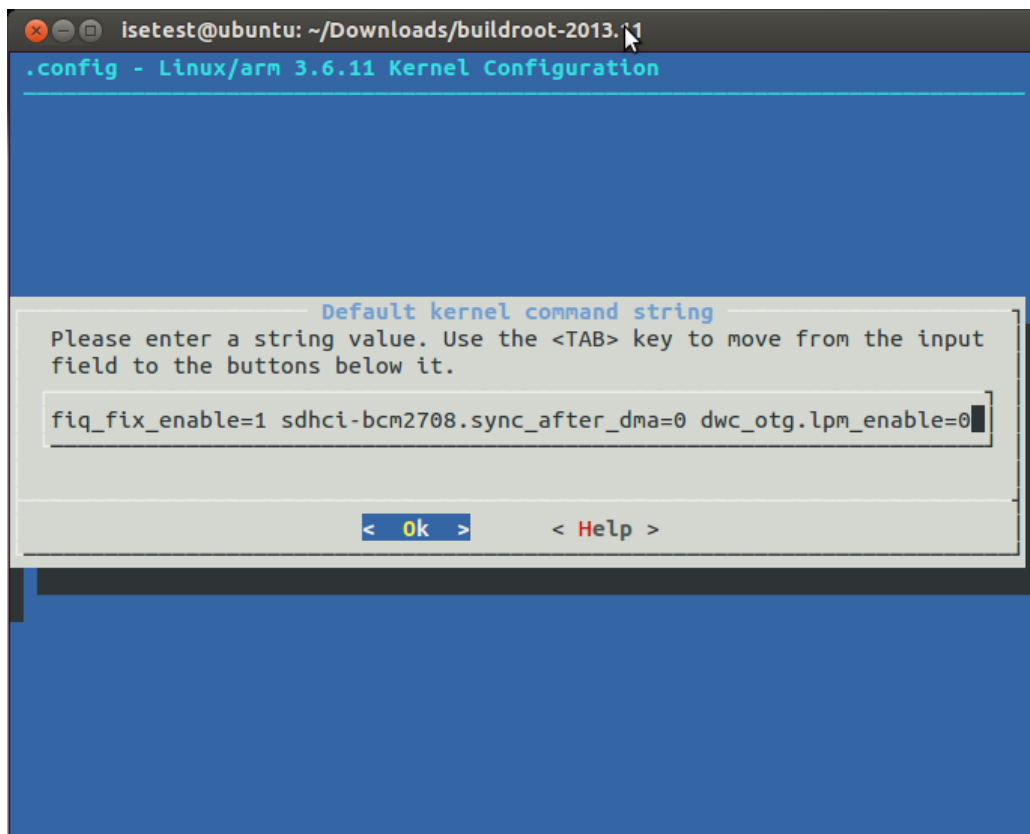


Fig. 14: Default argument to add.

Now, save the changes and execute the make command. You need to wait some minutes to finish the compilation of the kernel. Don't forget to copy the new images of the kernel and the filesystem to the SD card.

3.7 Booting the Raspberry Pi.

Fig. 15 displays a RaspBerry Pi. The description of this card, their functionalities, interfaces and connectors are explained in the ref [RD5]. The basic connection requires:

- a) To connect a USB to RS232 adapter (provided) to the raspberry-pi expansion header (see Fig. 16 and Fig. 17). This adapter will provide the serial line interface to be used as console in the Linux operating system.
- b) To connect the power supply with micro-USB connector provided (5 v).



Fig. 15: RaspBerry-Pi (Version B) hardware with main elements identified.

RASPBERRY PI Revision 2 Pinout

<http://www.pinballsp.com>

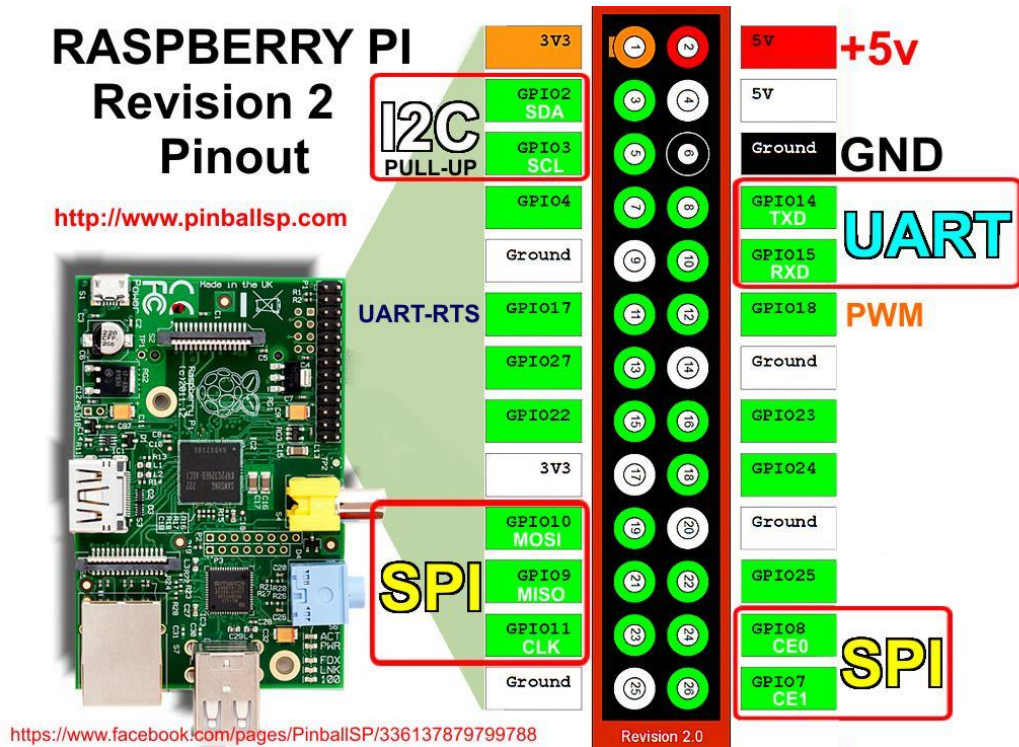


Fig. 16: Raspberry-PI header terminal identification.

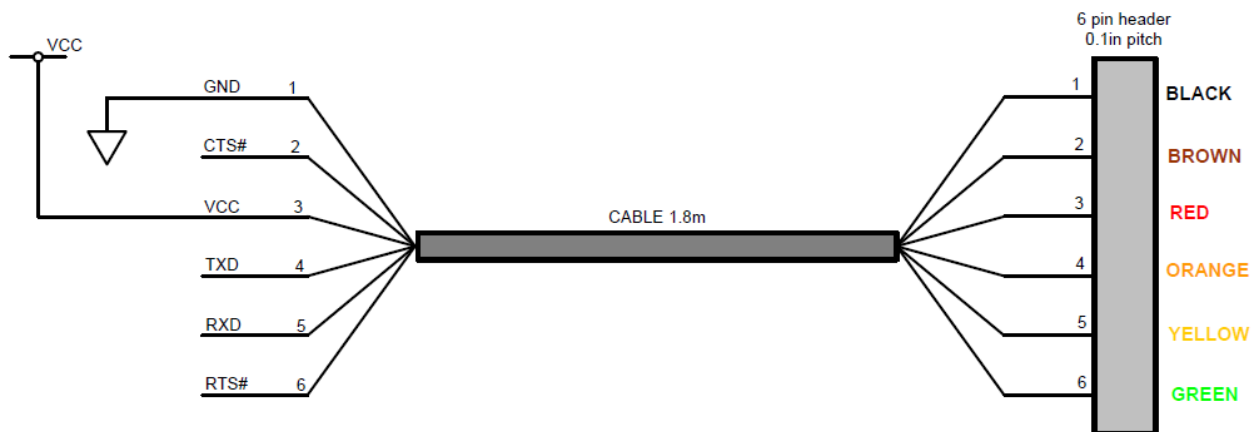


Fig. 17: Identification of the terminals in the USB-RS232 adapter

The booting process of the Raspberry Pi BMC2835 processor is depicted in Fig. 18. Take into account that this System On Chip (SoC), the BMC2835, contains two different processors. A GPU and an ARM processor. The programs bootcode.bin, loader.bin, start.elf are specifically written for GPU and the source code is not available. In fact Broadcom only provides details of this to customer that signs a commercial agreement. The last executable (start.elf) boots the ARM processor and allows the execution of programs written for ARM such as Linux OS.

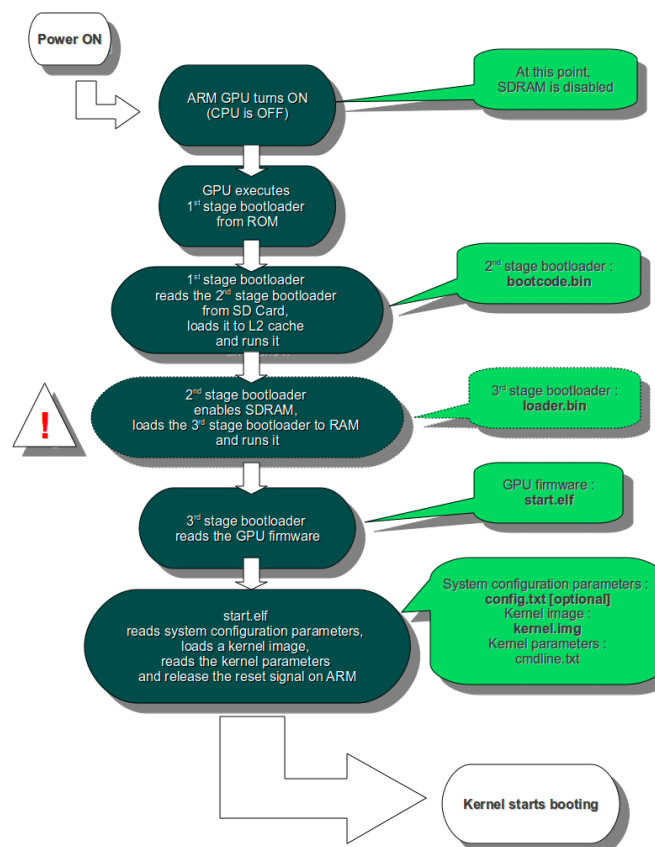


Fig. 18: Booting process for BMC2835 processor in the raspberry-pi.

By default the config.txt file contains these lines:

```

Please note that this is only a sample, we recommend you to change it to fit
# your needs.
# You should override this file using a post-build script.
# See http://buildroot.org/downloads/manual/manual.html#rootfs-custom
# and http://elinux.org/RPiconfig for a description of config.txt syntax

arm_freq=700
core_freq=250
kernel=zImage
disable_overscan=1
gpu_mem_256=100
gpu_mem_512=100
sdram_freq=400
over_voltage=0

```

Once the ARM has booted it executes the zImage application. This application is the Linux Kernel in zImage format (we have not used this type of image in our setup of Buildroot). The parameters that will be passed to the kernel are listed in the cmdline.txt file. For instance, by default, Buildroot generates this one:

```

dwc_otg.fiq_fix_enable=1 sdhci-bcm2708.sync_after_dma=0 dwc_otg.lpm_enable=0 console=tty1 root=/dev/mmcblk0p2
rootwait

```


In the Linux machine, or in the windows one, open a Terminal and execute the program putty (or hyperterminal), in a seconds a window will be displayed. Configure the parameters using the information displayed in Fig. 19 (for the specific case of putty), and then press “Open”. **You will see nothing the output.** Now, insert the SD card again in the host computer and edit the config.txt file modifying the line “kernel=zImage” by “kernel=u-boot.bin”. Boot again the RaspBerry-Pi and you will see u-boot boot messages (see Fig. 20).

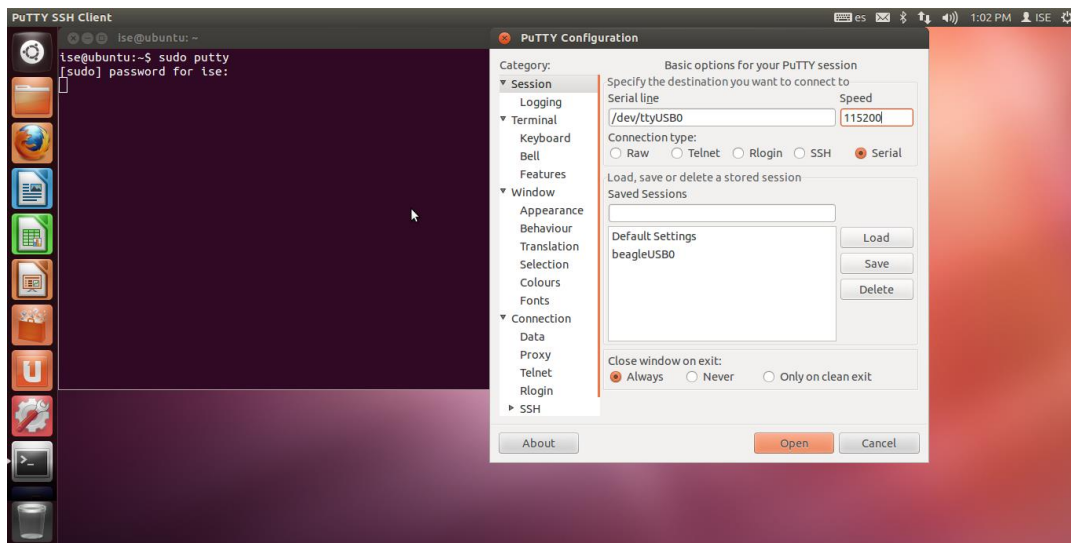


Fig. 19: Putty program main window.



[Serial interface identification in Linux]: In Linux the serial devices are identified typically with the names /dev/ttyS0, /dev/ttyS1, etc. In the figure the example has been checked with a serial port implemented with an USB-RS232 converter. This is the reason of why the name is /dev/ttyUSB0. In your computer you need to find the identification of your serial port. Use Linux dmesg command to do this.

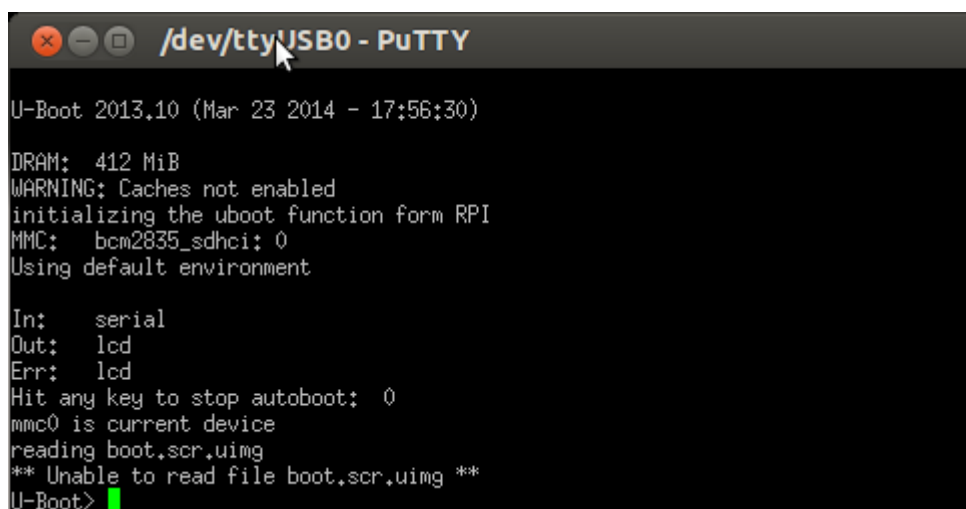


Fig. 20: Serial output in the Rasperry Pi boot process..

In this moment u-boot is running and you can introduce u-boot commands in order to start the execution of your system.



[Help]: If you obtain errors in uboot execution contact with your instructor to check possible issues.

Execute the following commands:

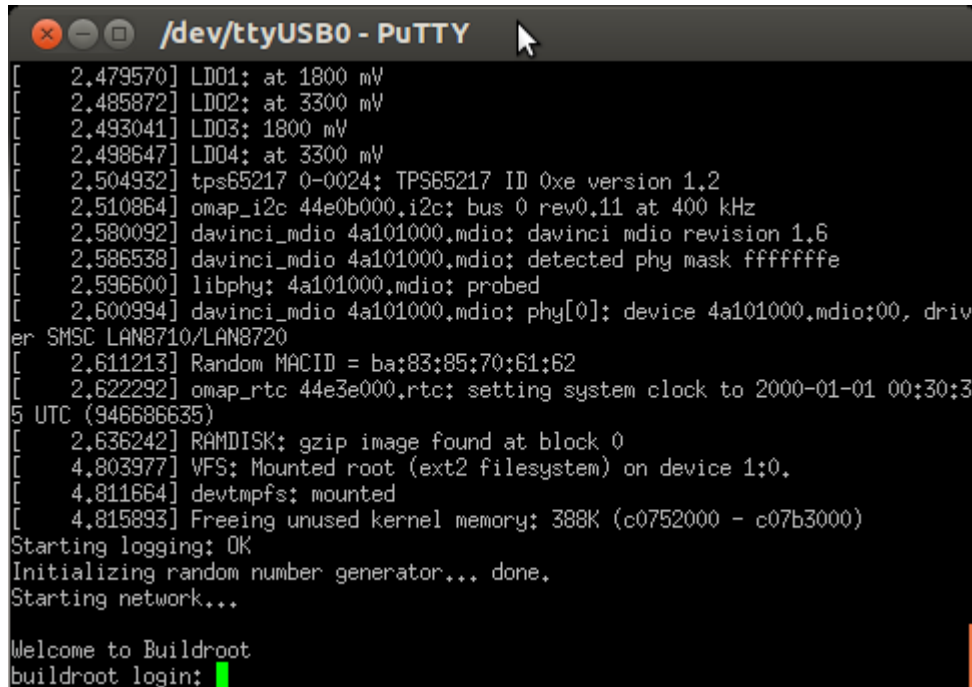
```
mmc rescan
mmc list
setenv bootargs console=ttyAMA0 root=/dev/ram rw ramdisk_size=16384 initrd=0x02000000,16M init=/sbin/init
fatload mmc 0 0x01000000 ulmage
fatload mmc 0 0x02000000 rootfs.ext2.gz
bootm 0x01000000
```

The meaning of the different command is explained in Table II.

Table II: u-boot parameters

Command	explanation
mmc rescan	Scan the availability of a MMC device
mmc list	List MC devices available in the system
setenv bootargs console=ttyAMA0,115200n8 root=/dev/ram rw ramdisk_size=16384 initrd=0x02000000,16M init=/sbin/init	Set the environment variables of u-boot. These variables are needed for a correct boot of Linux system Console is the serial line to display the boot messages (ttyAMA0 using 8 bits and 115200 baud rate) Root is the device containing the main partition to mount. Here we are using RAM memory to mount the filesystem Rw filesystem y read/write Ramdisk_size and initrd are parameters related with the size and location of the filesystem in the memory Init is the first process to execute in Linux (tip ically this) You can specify two different serial devices for the console: ttyAMA0 linked with the serial line and tty1 using VGA/HDMI as output and a USB keyboard as input.
fatload mmc 0 0x02000000 ulmage	Load from a MMC formatted as FAT32 the file ulmage in address 0x02000000. ulmage constains the kernel image.
fatload mmc 0 0x01000000 rootfs.ext2.gz	Load from a MMC formatted as FAT32 the file rootfs.ext2.gz in address 0x01000000. Rootfs.ext2.gz contains the file system compressed. This file will be uncompressed by Linux in the booting stage.
bootm 0x01000000	Boot the program starting at address 0x01000000. We are booting the kernel

After some seconds you will see a lot messages displaying in the terminal. Linux kernel is booting and the operating system is running its configuration and initial daemons. If the system boots correctly you will see an output like the represented in Fig. 21. Introduce the user name root and the Linux shell will be available for you.



```
[ 2.479570] LD01: at 1800 mV
[ 2.485872] LD02: at 3300 mV
[ 2.493041] LD03: 1800 mV
[ 2.498647] LD04: at 3300 mV
[ 2.504932] tps65217 0-0024: TPS65217 ID 0xe version 1.2
[ 2.510864] omap_i2c 44e0b000.i2c: bus 0 rev0.11 at 400 kHz
[ 2.580092] davinci_mdio 4a101000.mdio: davinci mdio revision 1.6
[ 2.586538] davinci_mdio 4a101000.mdio: detected phy mask ffffffff
[ 2.596600] libphy: 4a101000.mdio: probed
[ 2.600994] davinci_mdio 4a101000.mdio: phy[0]: device 4a101000.mdio:00, driv
er SMSC LAN8710/LAN8720
[ 2.611213] Random MACID = ba:83:85:70:61:62
[ 2.622292] omap_rtc 44e3e000.rtc: setting system clock to 2000-01-01 00:30:3
5 UTC (946686635)
[ 2.636242] RAMDISK: gzip image found at block 0
[ 4.803977] VFS: Mounted root (ext2 filesystem) on device 1:0.
[ 4.811664] devtmpfs: mounted
[ 4.815893] Freeing unused kernel memory: 388K (c0752000 - c07b3000)
Starting logging: OK
Initializing random number generator... done.
Starting network...

Welcome to Buildroot
buildroot login: █
```

Fig. 21: Beagle running Linux.

3.8 Booting the Raspberry Pi using a script.

The previous step aforementioned for starting up the Linux can be simplified a little bit using a u-boot script. If you create a text file with the u-boot commands you obtain a binary script using the following command:

```
mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "Raspberry Pi" -d boot_mmc.txt boot.scr.uimg
```

Copy the boot.scr.uimg into the SD card y reboot Raspberry Pi.

3.9 Configuring the network interface

In order to connect raspberry-pi to the network we need to set up the Ethernet interface and apply a IP address. The command to do this is:

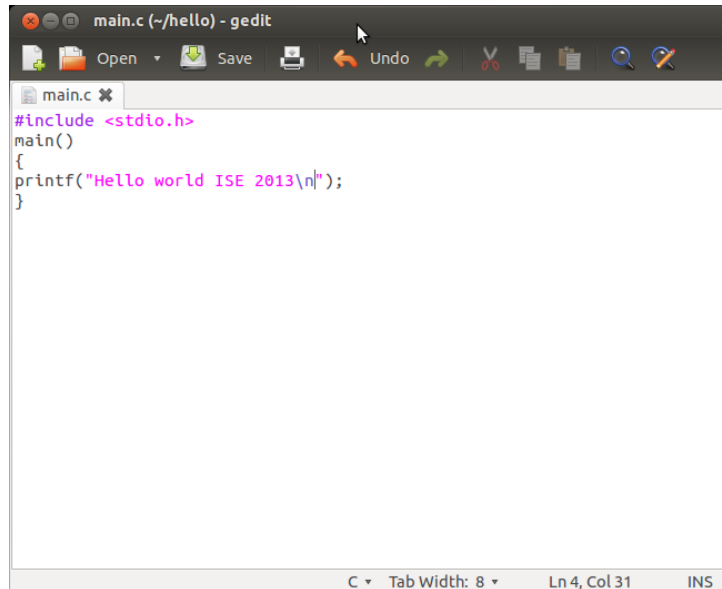
```
Ifconfig eth0 <ipaddress> netmask <netmask>
```

You need to know the IP network parameters. Request them to your instructor. Once the interface is up. You can test if the network is working using the ping command.

4 LAB2: CROSS-COMPILING APPLICATIONS FOR BEAGLEBOARD

4.1 Hello Word application.

The simplest application to be developed is the typical “Hello Word” application. Open a terminal in the Linux machine. Create a folder named “hello”, change the directory to it and execute “gedit main.c” edit the program and save it.



```
main.c (~/hello) - gedit
#include <stdio.h>
main()
{
printf("Hello world ISE 2013\n");
}
```

Fig. 22: Basic hello world program in C.

The next step is to compile the program. Remember that we are developing cross applications. We are developing and compiling the code in a Linux x86 machine and we are executing it in an ARM architecture (see Fig. 23).

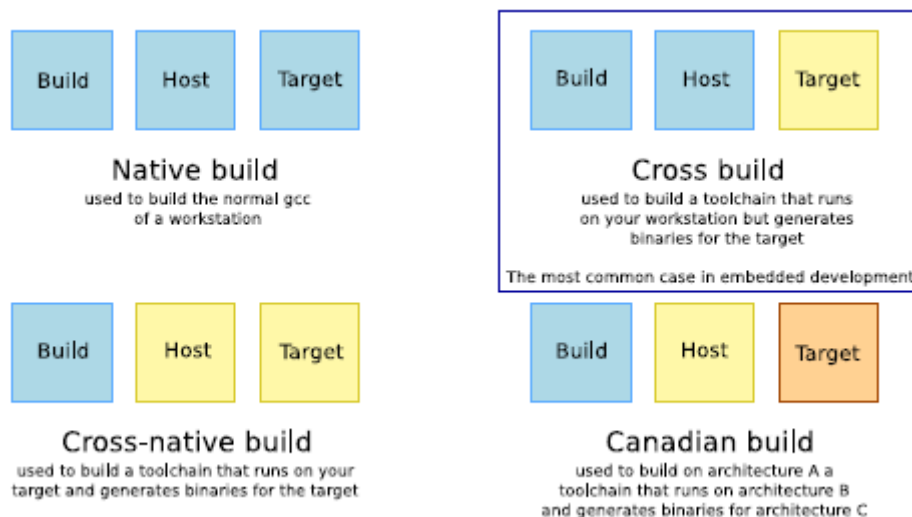


Fig. 23: Summary of the different configurations for developing applications for embedded systems. Figure copied from “Free Electrons” training materials (<http://free-electrons.com/training/>)

The first question is where the cross-compiler is located. The answer is this: in the folder “buildroot-2013.11-rc1/output/host/usr/bin”. If you inspect the content of this folder you can see the entire

compiling, linking and debugging tool (see Fig. 24). These programs can be executed in your x86 computer but will generate code for ARM processor.

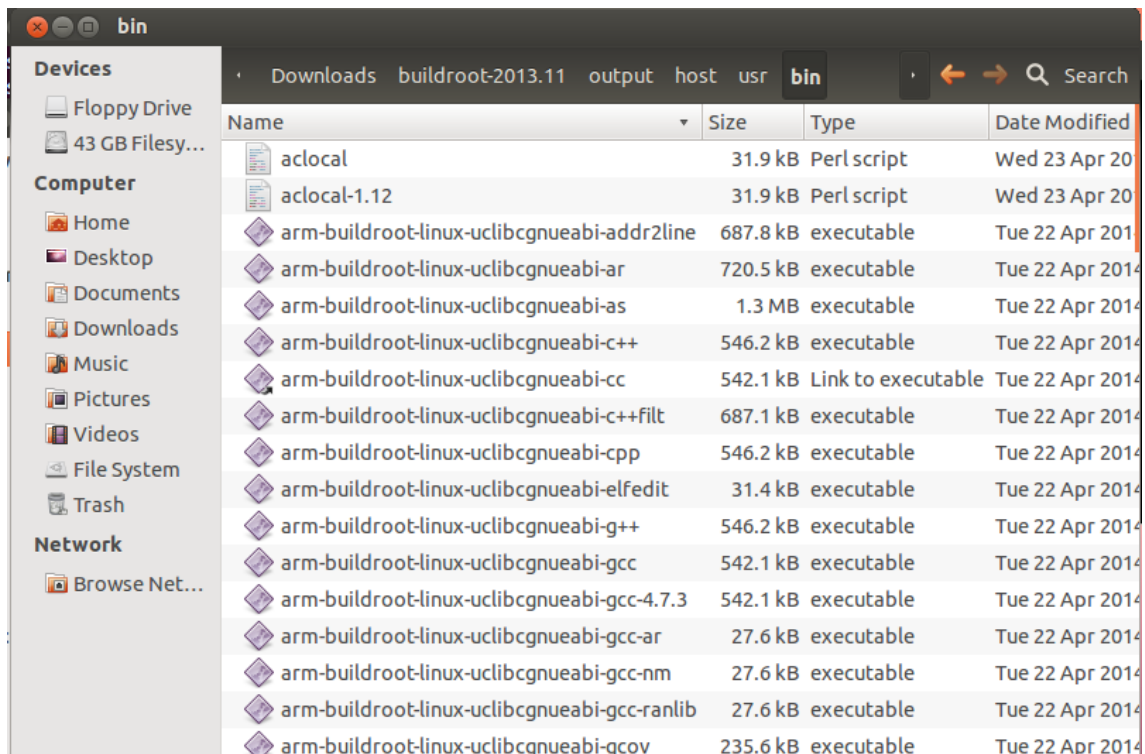
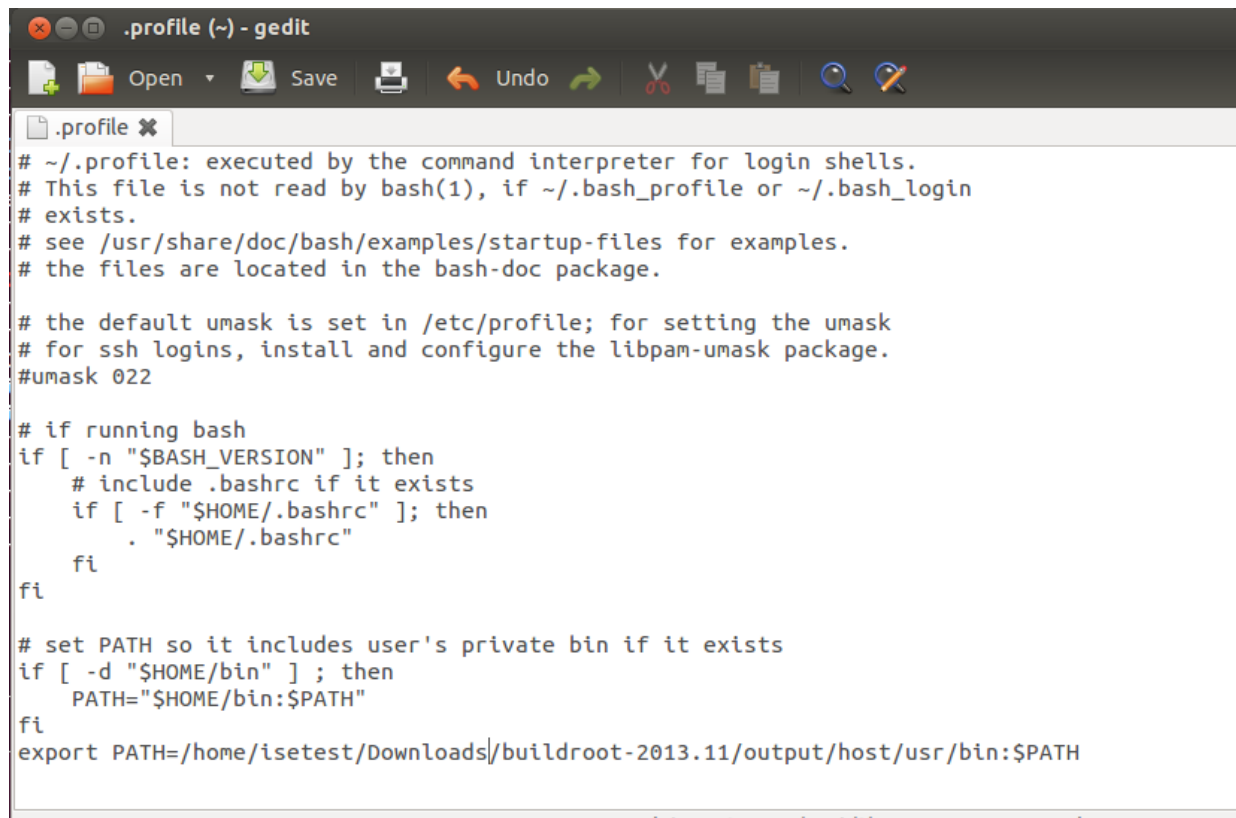


Fig. 24: Folder containing the cross compiling tools.

The second question is how to invoke the command to compile the code. Typically in Linux we use gcc command and some parameters. Previously to answer this question we are going to add the location of the cross-compiling tools to the Linux path. To do that you need to edit in your home directory the “.profile” file and add the corresponding path (see Fig. 25). Once you have edited the file you need to logout and login again.



[Changes in .profile]: changes in .profile file only have effect in you **execute a logout and, login again** into your account.



```
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile; for setting the umask
# for ssh logins, install and configure the libpam-umask package.
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
export PATH=/home/isetest/Downloads/buildroot-2013.11/output/host/usr/bin:$PATH
```

Fig. 25: Adding the cross compiler PATH to the Linux .profile file.

In this point we can compile and execute our first cross-compiled program. Execute this command:

```
arm-linux-gcc -o main main.c
```

Now move the main object program to the target. You need to copy the program using utilities like “scp”



[scp]: Use Linux documentation to see how to use scp command.

5 LAB3: USING INTEGRATED DEVELOPMENT ENVIRONMENT: ECLIPSE/CDT

5.1 Cross-Compiling application using Eclipse.

In a Terminal window start Eclipse with the following command:

```
ise@ubuntu:~$ eclipse
```

The popup window invites you to enter the workspace (see Fig. 26). The workspace is the folder that will contain all the eclipse projects created by the user. You can have as many workspaces as you want.

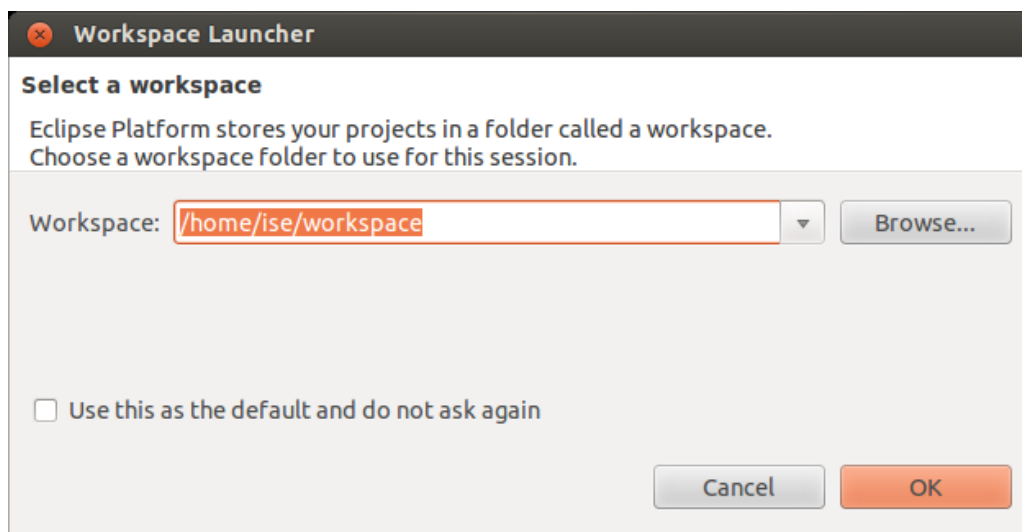


Fig. 26: Selection of the workspace for Eclipse.

Select Ok and the main window of Eclipse will be shown. Next close the welcome window and the main eclipse window will be displayed.

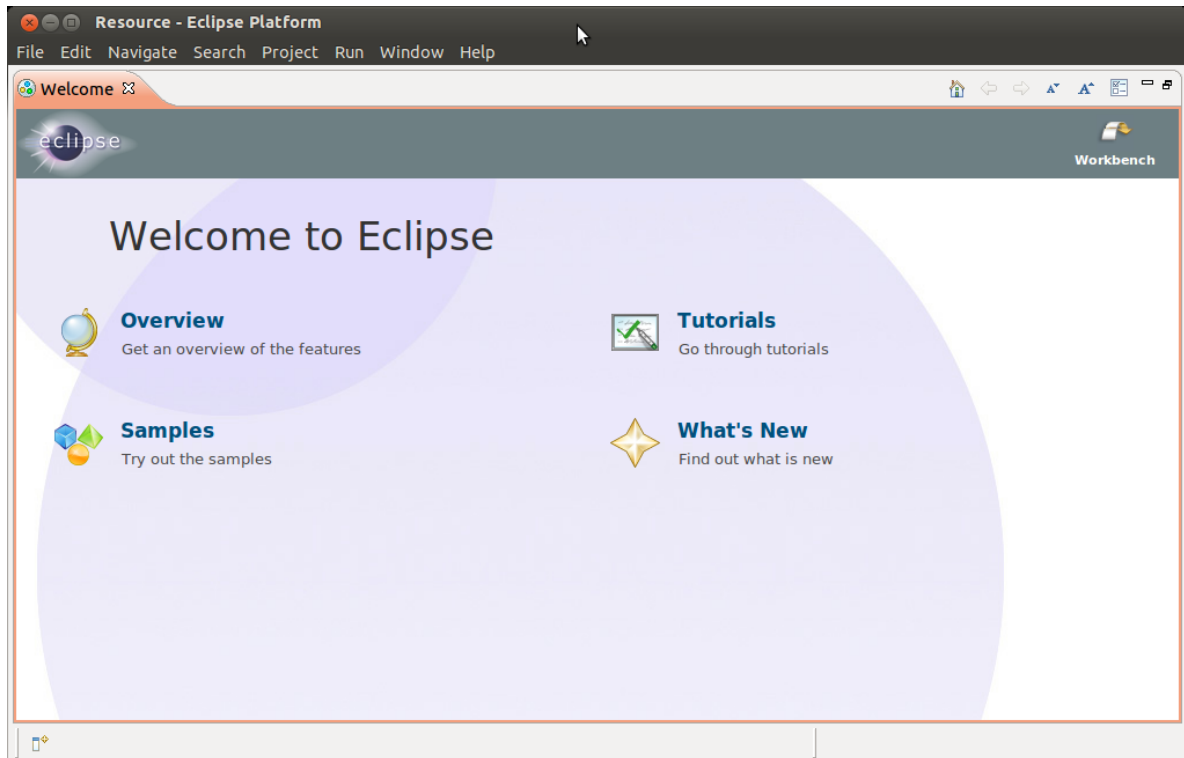


Fig. 27: Eclipse welcome window.

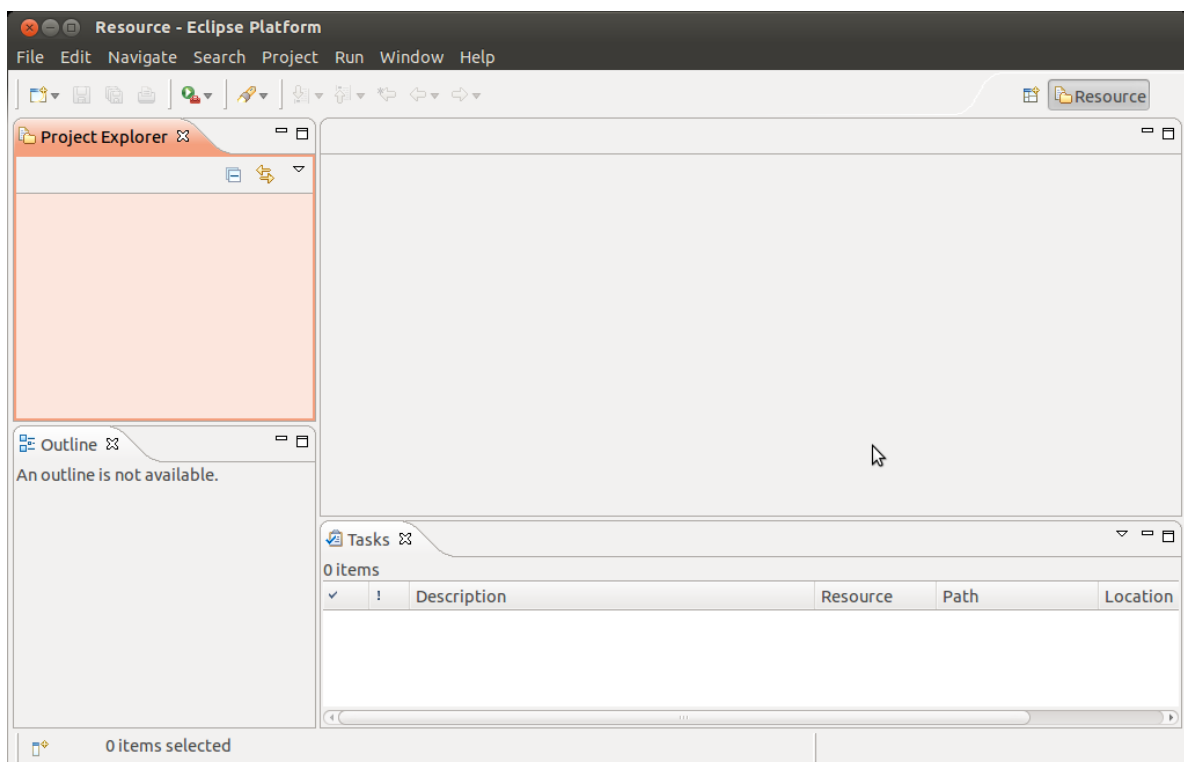


Fig. 28: Eclipse main window.

Create an Eclipse C/C++ project (File->New->Project) selecting the hello world example (see Fig. 29 y Fig. 30).

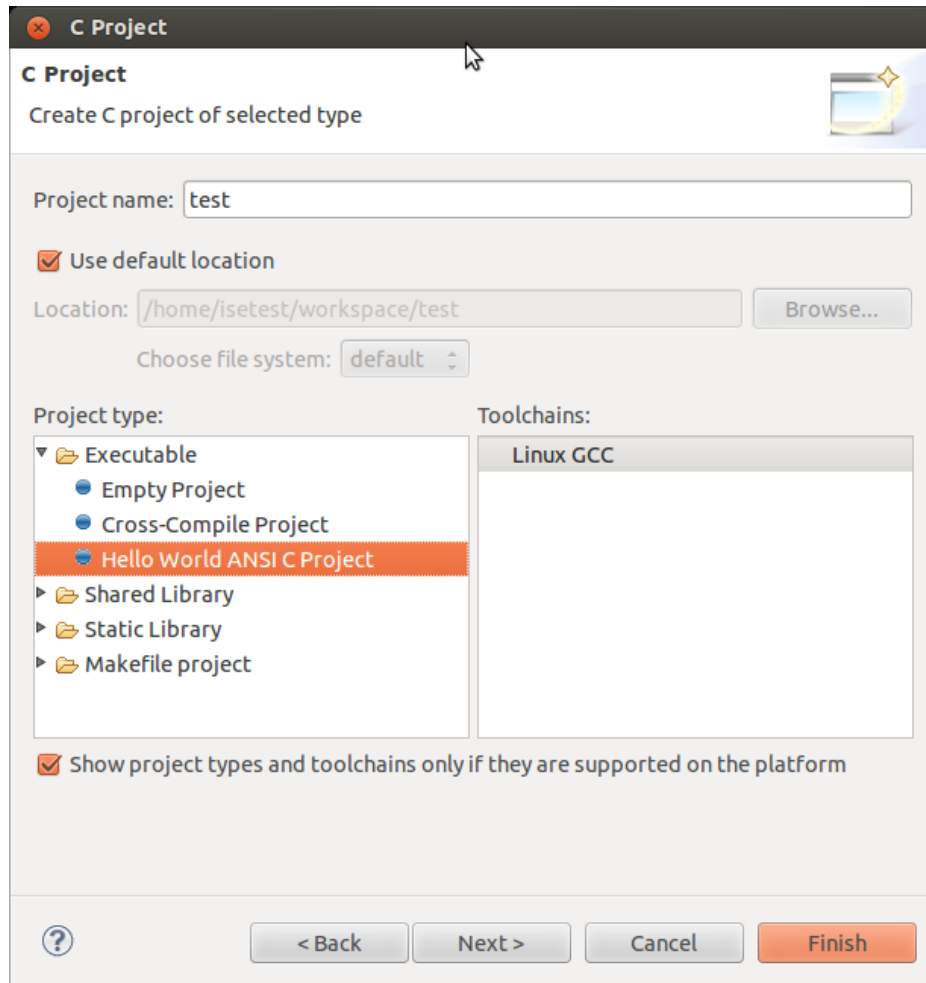


Fig. 29: Creation of the hello world C project.

Click on the Finish button and you will obtain your first project created with eclipse.

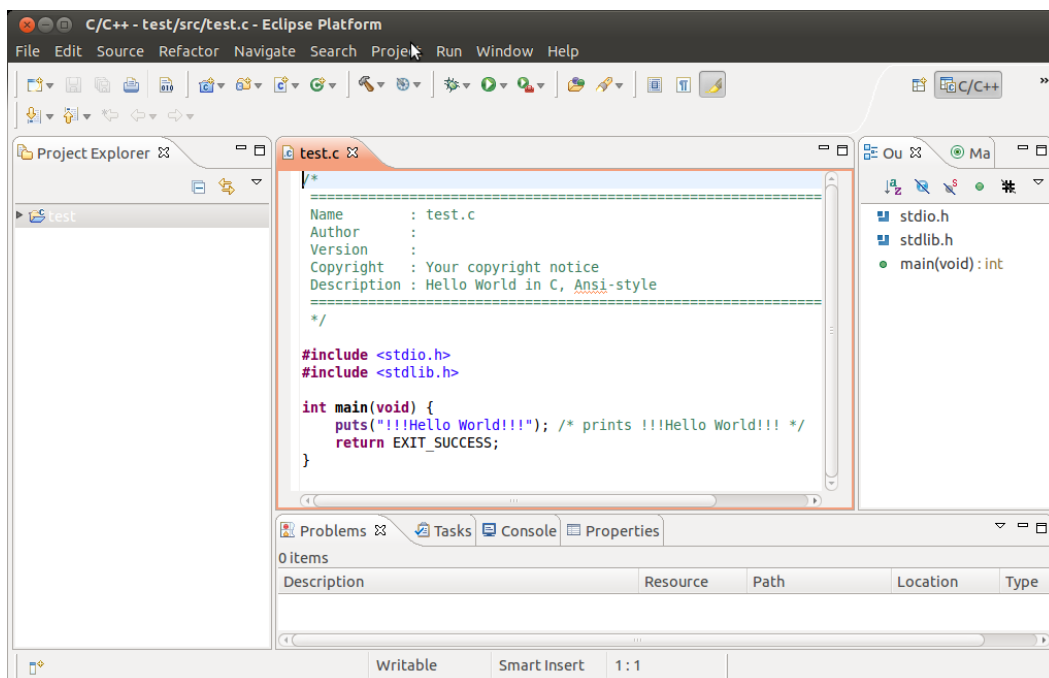


Fig. 30:Hello world example.

The next step is to configure the Eclipse project for managing the Cross-tools. In project ->properties configure the C/C++ Build Setting as the Fig. 31 and Fig. 32shown.

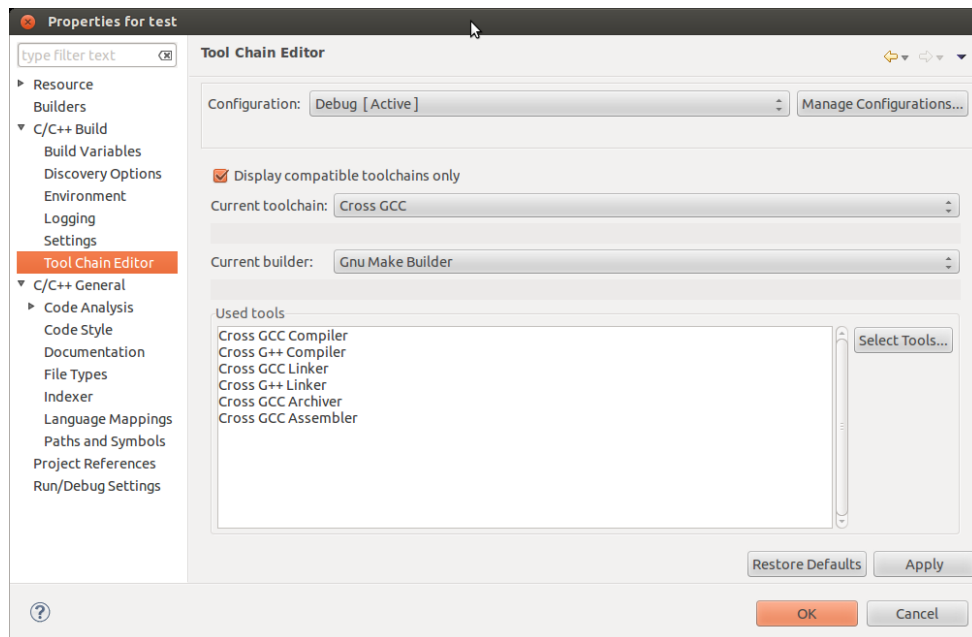


Fig. 31: Tool Chain Editor should be configured to use Cross GCC.

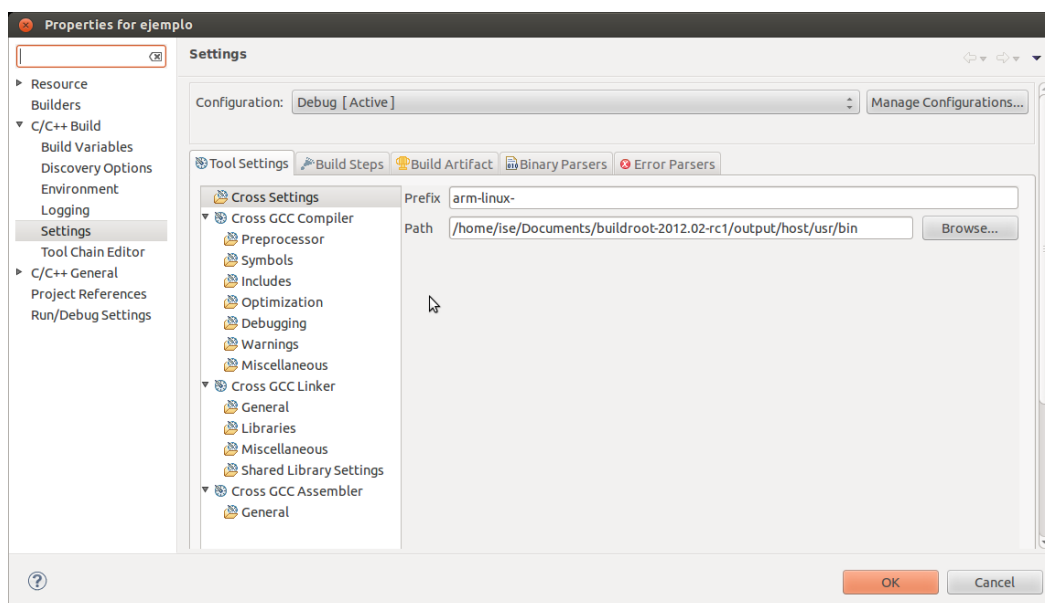


Fig. 32: Cross tools locate on (path)

The next step is to configure the search paths for the compiler and linker, and the different tools to use. Complete the different fields with the information included in Fig. 33 and Fig. 34.

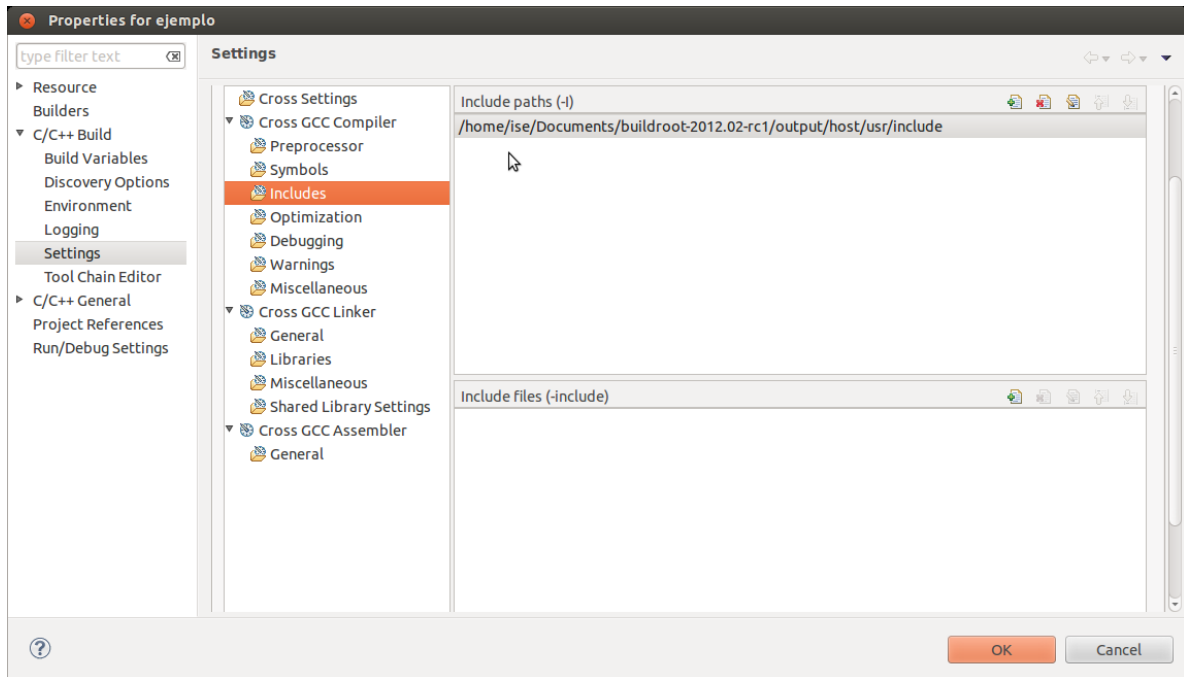


Fig. 33: Include search path.

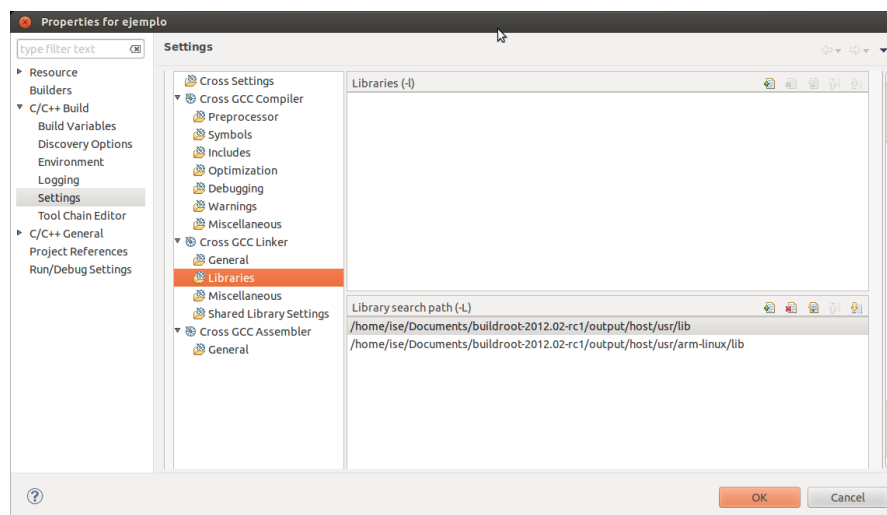


Fig. 34: Libraries search path.

Once you have configured the cross chain in Eclipse you can build your project using **Project->Build Project**. If everything is correct you will see the eclipse project as represented in Fig. 35.

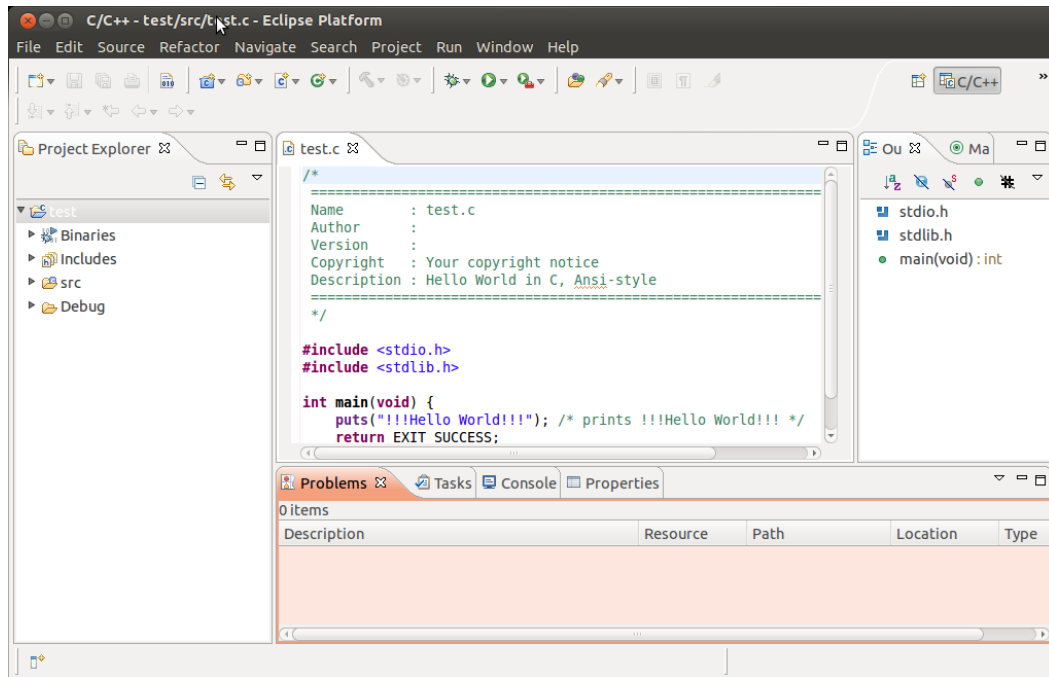


Fig. 35: Eclipse project compiled (Binaries has been generated).

Copy the executable program to the Raspberry-PI and execute the program using the gdbserver with this command:

```
ise@ubuntu:~$ gdbserver <ipaddresshostcomputer>:2345 ./main
```

Now, it is time to setup the debugging session in Eclipse environment. Select Run->Debug Configurations and complete the different tabs following the indications of Fig. 36, Fig. 37, Fig. 38 and Fig. 39

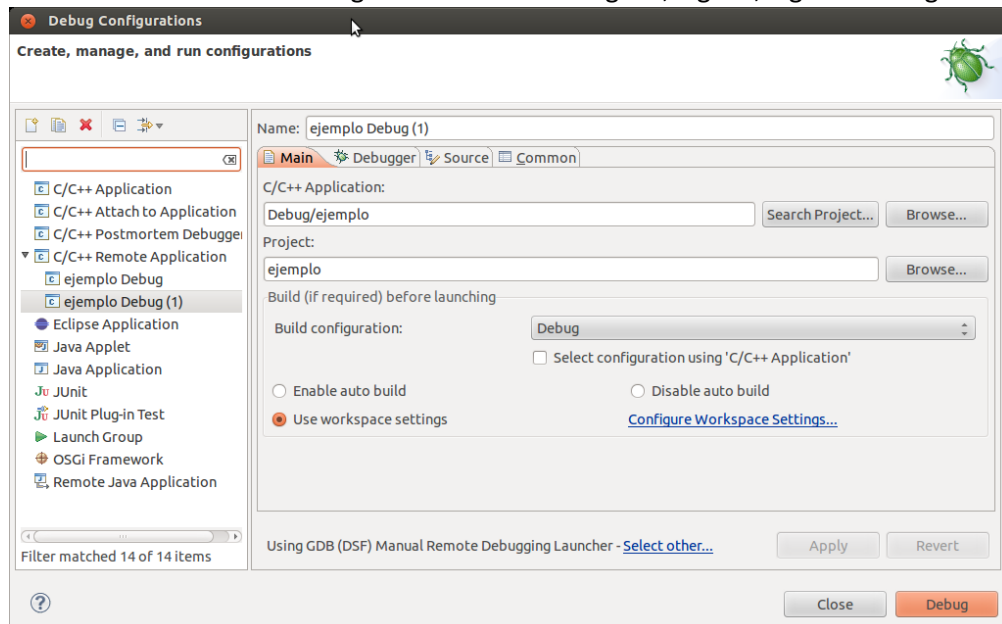


Fig. 36: Creating a Debug Configuration

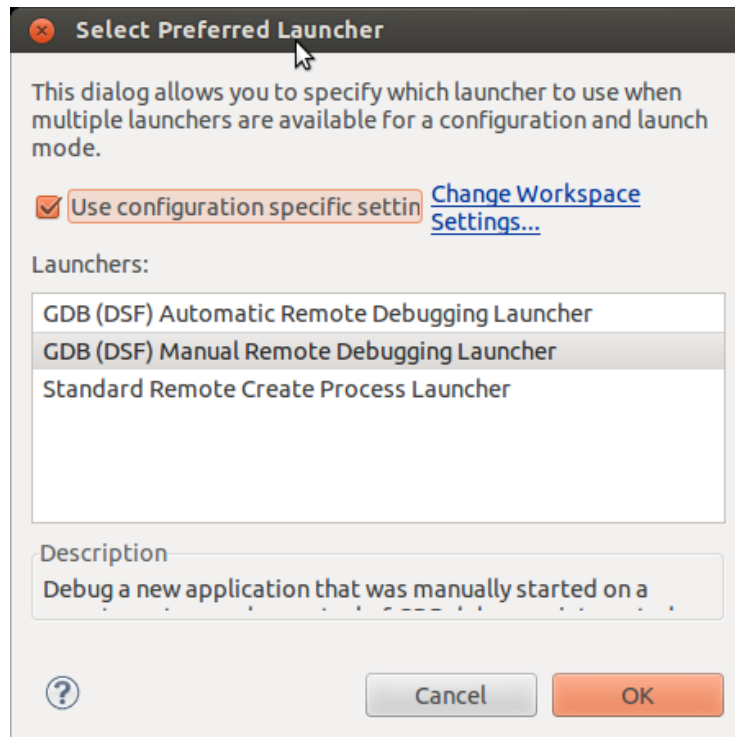


Fig. 37: Configuration must be set to manual remote debugging.

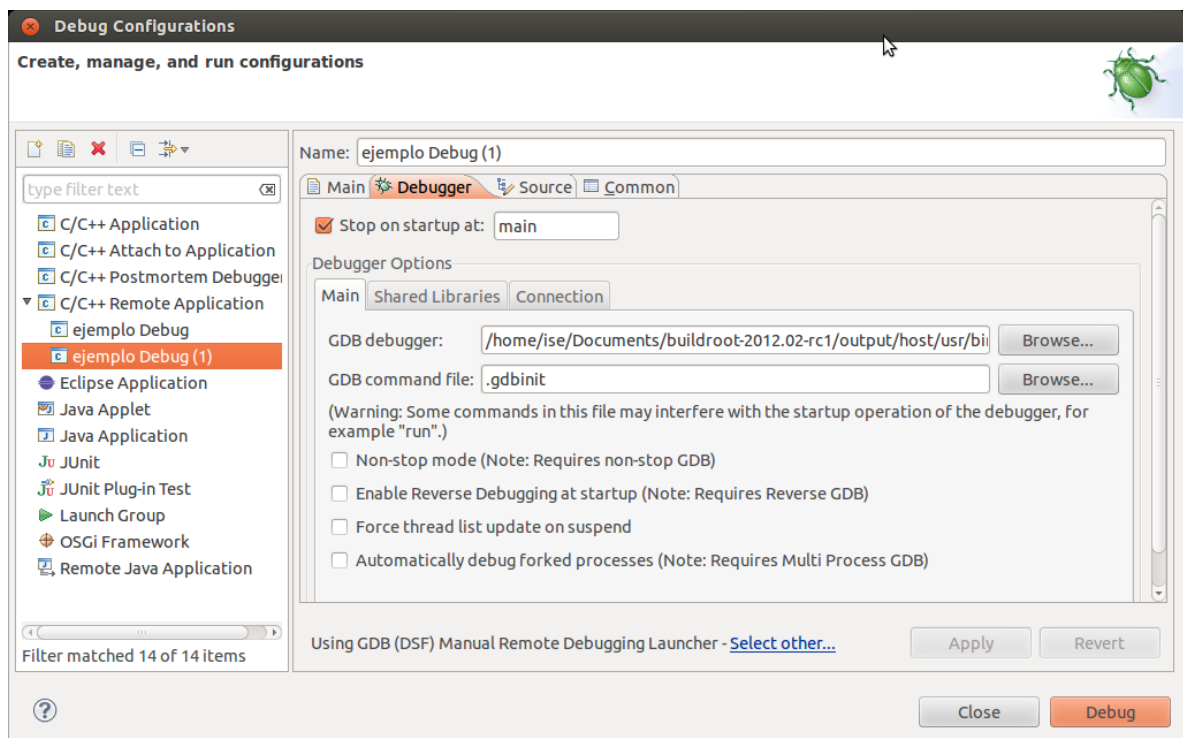


Fig. 38: Debug configuration including the path to locate the cross gdb tool.

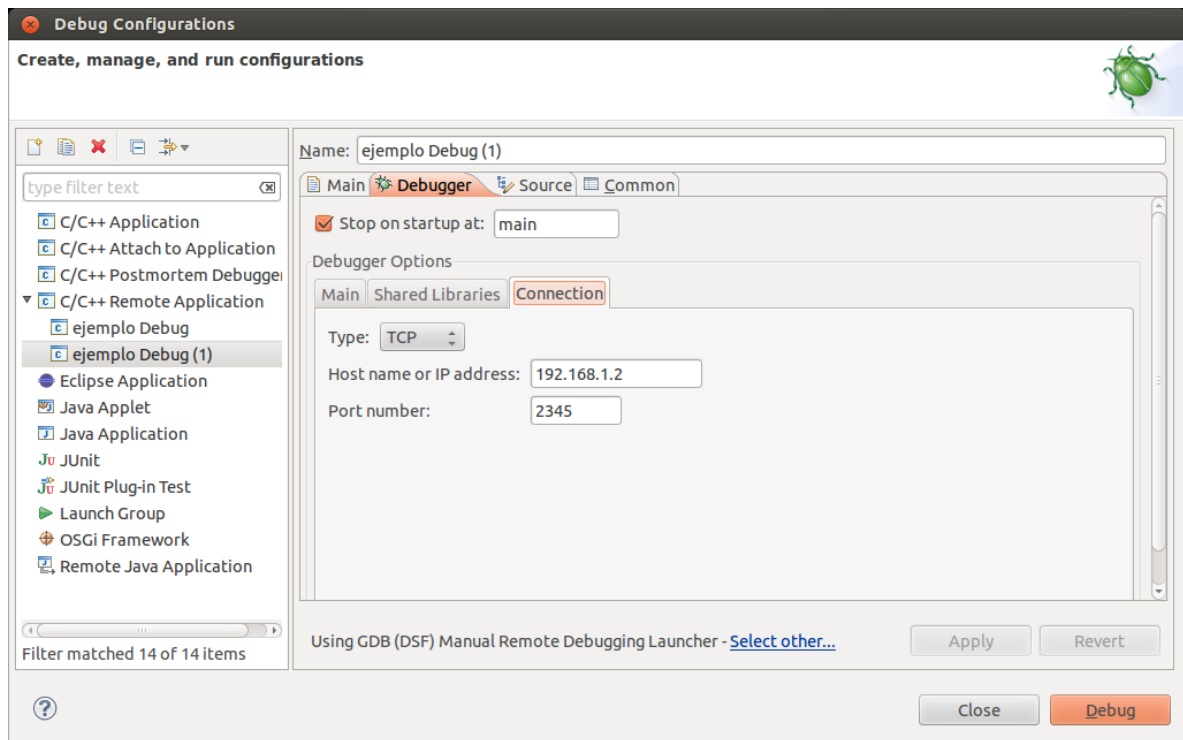


Fig. 39: Configuration of the remote target. Port number must be the same in the target and in the host.

Now, press Debug in Eclipse window and you can debug remotely your application.

6 PREPARING THE LINUX VIRTUAL MACHINE.

6.1 Download VMware Player.

The document <http://www.vmware.com/pdf/VMwarePlayerManual10.pdf> describes the installation and basic use of VMware Player. Follow the instructions to setup the application in your computer.

6.2 Installing Ubuntu 12.04 LTS as virtual machine.

The first step is to download Ubuntu 12.04 from Ubuntu web site using this link:

<http://www.ubuntu.com/download/desktop/thank-you?release=its&bits=32&distro=desktop&status=zeroc> . You will download an ISO image with this Linux operating System.

Run VMware player and install Ubuntu using the VMWare player instructions. A tutorial explaining this can be found here (<http://www.computersecuritystudent.com/UNIX/UBUNTU/lesson1/>)

6.3 Installing packages for supporting Buildroot.

The annex I contains the instructions for downloading the list of packages installed in the Ubuntu 12.04 LTS in order to run correctly Buildroot tools.

7 ANNEX I: UBUNTU 12.04 LTS PACKAGES INSTALLED.

7.1 List of packages installed in the Ubuntu OS.

In the moodle site there is a test file with a list of all the packages installed in the Ubuntu OS.

7.2 Installing software in Ubuntu 12.04 TLS

If you download that file you can replicate the installation using these commands:

```
sudo apt-get update
sudo apt-get dist-upgrade
sudo dpkg --set-selections < iseubuntu-files
sudo dselect install
```